Castro, Simão Pedro Pereira

Análise de imagens médicas com recurso a metodologias de deep learning

http://hdl.handle.net/11067/5979

Metadados

Data de Publicação 2021

Resumo A imagiologia médica refere-se a um conjunto de processos ou técnicas que permitem criar representações visuais das partes interiores do corpo. A avaliação de uma imagem médica requer uma análise cuidadosa bem como a compreensão das propriedades e dos detalhes das imagens, que incluem as condições de aquisição, as condições experimentais e as características do sistema biológico. O recurso à imagiologia médica permite a investigação e o diagnóstico precoce de diferentes patologias. Portanto, uma...

Medical imaging encompasses a set of processes or techniques which allow the creation of visual representations of the inner parts of the body. The evaluation of a medical image requires a careful analysis, as well as the understanding of the properties and details of the images, that include the acquisition and experimental conditions, and the features of the biological system. The use of medical imaging allows the investigation and the early diagnosis of different pathologies. Therefore, a kno...

Palavras Chave Engenharia, Diagnóstico assistido por computador, Deep learning

Tipo masterThesis

Revisão de Pares no

Coleções [ULF-FET] Dissertações

UNIVERSIDADE LUSÍADA – NORTE
***CAMPUS* DE VILA NOVA DE FAMALICÃO**

ANÁLISE DE IMAGENS MÉDICAS COM RECURSO A METODOLOGIAS DE *DEEP LEARNING*

Simão Pedro Pereira Castro

Orientador: Professor Doutor Rui Silva

Dissertação para obtenção do Grau de Mestre em Engenharia Eletrónica e Informática

Agradecimentos

Cinco anos depois, termina um ciclo, um sonho há muito tempo desejado. Percorrer este caminho só foi possível com o apoio, carinho e força das pessoas que me rodeiam. O meu muito obrigado a todos aqueles que de algum modo contribuíram para o desenvolvimento e conclusão deste trabalho/ciclo. No entanto, gostaria de expressar a minha mais profunda gratidão particularmente ao:

Professor Doutor Rui Silva, orientador desta dissertação, pelos valiosos ensinamentos, espírito crítico notável, todo o incentivo e amizade demonstrados ao longo deste trabalho. Um especial obrigado por toda a disponibilidade.

Professor Doutor Vítor Pereira, pelos ensinamentos transmitidos ao longo de todo este percurso, bem como pelo incentivo constante.

A todos os professores da Faculdade de Engenharia da Universidade Lusíada *campus* de Famalicão, assim como à instituição.

À Joana Moreira, o meu muito obrigado pelo carinho, paciência, dedicação e motivação constante. Foste incansável, e este percurso, muito se deve a ti.

Aos meus pais, irmãos, sogros e restante família, obrigado pela confiança, paciência e apoio sem igual. Ao Fred e a Lu, um especial obrigado pela amizade, motivação e confiança dada ao longo de todo este caminho.

A todos os meus amigos de longa data e às amizades nascidas na faculdade, pelo apoio no estudo, pelos momentos de alegria, pela paciência e pela motivação.

Aos amigos do trabalho, pela paciência e por me ajudarem com todas as trocas de horários e, de modo particular, aos Engenheiros Filipe Costa, Roberto Araújo, Cláudia Vieira, João Correia, Vasco Mendes e ao Adelino Ramos.

"If I have seen further, it is by standing upon the shoulders of giants."

Sir Isaac Newton

Resumo

A imagiologia médica refere-se a um conjunto de processos ou técnicas que permitem criar representações visuais das partes interiores do corpo. A avaliação de uma imagem médica requer uma análise cuidadosa bem como a compreensão das propriedades e dos detalhes das imagens, que incluem as condições de aquisição, as condições experimentais e as características do sistema biológico. O recurso à imagiologia médica permite a investigação e o diagnóstico precoce de diferentes patologias. Portanto, uma abordagem baseada no conhecimento para a análise e interpretação de tais imagens é imperativa.

Há cada vez mais inovações no que concerne ao diagnóstico através de imagens médicas. Como tal, os avanços técnicos que permitam a produção de imagens de maior resolução, aliados a métodos de análise de imagens médicas que permitam extrair novas informações, têm sido investigados por parte da comunidade científica. Uma das áreas de investigação em destaque consiste na aplicação da inteligência artificial na imagem médica emulando a racionalidade do diagnóstico realizada pelo médico e oferecendo uma oportunidade para novos desenvolvimentos no que concerne à utilização da imagem médica como ponto de partida para o diagnóstico.

Este trabalho visa investigar e implementar metodologias de *machine learning*, um ramo da inteligência artificial, para classificar e segmentar imagens médicas. Para tal, foi realizada uma extensa pesquisa bibliográfica sobre o estado da arte em revistas da especialidade indexadas. No sentido de testar diferentes abordagens foram selecionados para teste dois dataset para classificação, *MedMNIST* e *MedNIST* compostos por 454591 e 58954 imagens médicas respetivamente, e dois dataset para segmentação, *BBBC038* composto por 735 imagens médicas e o *ICPR2012* com 50 imagens H&E.

Assim, o trabalho foi dividido em duas vertentes principais. Uma primeira parte onde se foca na classificação de imagens médicas, onde foi implementada e comparada a performance de várias arquiteturas utilizando as métricas adequadas. Para a realização desta primeira tarefa, foi necessário um pré-processamento dos dados (das imagens médicas). Em segundo lugar, foram investigadas formas de segmentação de imagens com o intuito de identificar núcleo celulares. Uma vez mais, foram construídas e comparadas as performances de diferentes arquiteturas, utilizando as métricas mais pertinentes. Adicionalmente, foi investigada a segmentação e a deteção com a particularidade de identificar núcleos que se encontrassem em mitose. Para ambas as tarefas foram obtidos resultados mais promissores do que os previamente reportados para os dataset's estudados. No final, foi ainda desenvolvida uma aplicação web que permite testar os modelos e visualizar os resultados.

Em resumo, os resultados deste estudo demonstraram o potencial das metodologias de *machine learning* como uma ferramenta importante para automatização de tarefas na área de imagem médica

apresentando importantes contributos que permitem uma melhoria na classificação de determinadas patologias.

Palavras-Chave: Imagem médica; Diagnóstico assistido por computador; *Machine learning*; *Deep learning*; Classificação; Segmentação; CNN.

Abstract

Medical imaging encompasses a set of processes or techniques which allow the creation of visual representations of the inner parts of the body. The evaluation of a medical image requires a careful analysis, as well as the understanding of the properties and details of the images, that include the acquisition and experimental conditions, and the features of the biological system. The use of medical imaging allows the investigation and the early diagnosis of different pathologies. Therefore, a knowledge-based approach for the analysis and interpretation of such images is imperative.

There is an increasing innovation concerning diagnosis through medical imaging. As such, the technical advances that allow the production of higher resolution images, allied to methods of medical images analysis that uncover new information, have been investigated by the scientific community. A research field that must be highlighted within medical imaging is artificial intelligence, which emulates the rationality of the diagnosis performed by the medical doctor and offers an opportunity for new developments regarding the use of medical imaging as a starting point for diagnosis.

This work aims to investigate and implement machine learning methodologies, a field of artificial intelligence, to classify and segment medical images. For that goal, an intensive literature search in indexed specialty journals was conducted. As a way to test different approaches, two datasets were selected for classification, *MedMNIST* and *MedNIST*, composed by 454591 and 58954 medical images, respectively, and two datasets for segmentation, *BBBC038*, composed by 735 medical images and *ICPR2012* with 50 H&E images.

Therefore, this work was divided into two main components. A first part, where the focus is on the classification of medical images, where the performance of several architectures was implemented and characterized, using the adequate metrics. To accomplish this first task, a pre-processing of the data (medical images) was needed. Secondly, the segmentation of images with the goal of identifying cell nuclei were investigated. Once again, the performances of several architectures were built and compared, using the most relevant metrics. Additionally, research was conducted concerning segmentation and detection, with the singularity of identifying nuclei undergoing mitosis. The results obtained were more promising for both tasks than what had previously been reported for the studied datasets. In the end, a web application capable of testing the models and visualize the results was developed.

In brief, the results obtained herein demonstrate the potential of machine learning methodologies as an important tool for the automatization of tasks in the medical imaging field, providing important contributions that lead to a better classification of certain pathologies.

Key Words: Medical Image; Computer-aided diagnostics; Machine Learning; Deep learning; Classification; Segmentation; CNN.

Índice

Índice de Figuras

Índice de Tabelas

Lista de Abreviaturas

ACC	Exatidão
ADASYN	Amostragem sintética adaptativa
ADALINE	Neurónio linear adaptativo
AE	Autoencoder
ANN	Redes neuronais artificiais
AUC	Área sob a curva ROC
BCE	Entropia cruzada binária
BD	Bloco denso
BPTT	*Back Propagation Through Time*
CapsNet	*Capsule Network*
CE	Entropia cruzada
CNN	Rede neuronal convolucional
CONV	Camadas convolucionais
DAE	*Denoising*
DBN	*Deep Belief Network*
DCNN	*Redes neuronais convolucionais profundas*
DL	*Deep learning*
DNN	*Deep neural network*
DT	Árvores de decisão
ECG	Eletrocardiograma
EEG	Eletroencefalografia
ELUs	*Exponencial linear units*
EN	*EfficientNet*
FC	Camada totalmente ligada
FCL	Camada de ligação completa
FFNN	Rede neuronal *feed-forward*
FN	Falsos negativos
FP	Falsos positivos
FPS	Espectro de potência de F*ourier*
GAN	*Generative Adversial Networks*

GIA	Angiectasia gastrointestinal
GLCM	Matriz de ocorrências de níveis de cinzentos
IoU	Interseção sobre a união
KNN	*K-Nearest neighbor*
KPCA	*Kernel* PCA
LReLU	*Leaky* ReLU
LSTM	*Long Short Term Memory*
MEG	Magnetoencefalografia
ML	*Machine learning*
MRDM	*Multi relational data mining*
MRF	*Markov Random Field*
MRI	Ressonâncias magnéticas
MSE	Erro médio quadrático
NB	*Naïve Bayesian*
OMS	Organização mundial de saúde
PCA	Análise de componentes principais
PELU	*Parametric exponencial linear unit*
POOL	camadas de *pooling*
PReLU	*Parametric rectified linear units*
RBM	*Boltzmann*
RDM	*Relacional data mining*
ReLU	*Rectified linear unit*
RL	*Reinforcement learning*
RNN	Rede neuronal recorrente
ROI	região de interesse
RReLU	*Randomized leaky* ReLU
SAE	Autocodificador empilhado
SELU	*Scaled exponencial linear units*
SGD	*Stochastic gradient descent*
SMOTE	Amostragem excessiva de minoria sintética
Sn	Sensibilidade
Sp	Especificidade

SPECT	Tomografia computorizada por emissão de fotões
SVM	Máquinas de suporte vetorial
TAC	Tomografia computorizada axial
Tanh	*Hyperbolic tangent*
TN	Verdadeiros negativos
TP	Verdadeiros positivos
TSF	Filtro não linear com estrutura em árvores
TSWT	Transformadores de onda de árvore estruturada
VAE	Autoencoder variável
VGG	*Visual Geometric Group*
WCE	Endoscopia com recurso a cápsulas sem fios
WPT	Transformações *wavelet* por pacotes

Capítulo 1: Introdução

Neste capítulo é apresentada uma breve contextualização do trabalho e as motivações subjacentes ao seu desenvolvimento. Os principais objetivos são ainda inumerados e é apresentada uma breve descrição da metodologia aplicada. Este capítulo termina com uma breve explicação da estrutura da presente dissertação.

1.1 Contexto e Motivação

A imagiologia médica tem um papel de extrema relevância no diagnóstico e tratamento de patologias. A qualidade destas imagens assim como o seu processamento melhora a tomada de decisões médicas permitindo a deteção de patologias de modo precoce e até reduzir custos nos tratamentos posteriores.

Apesar do sucesso no diagnóstico realizado por especialistas recorrendo a imagem médica, a análise e interpretação automática de imagens representa um avanço significativo no apoio aos especialistas e simultaneamente potencia a identificação precoce e massificada de alguns tipos de doenças. Diversos estudos têm sido propostos na literatura para melhorar o diagnóstico, assim como para solucionar problemas específicos em diferentes modalidades de imagem médica recorrendo a estudos computacionais sendo, no entanto, requerida a sua otimização por forma a que os seus resultados sejam generalizadamente aceites. Por conseguinte é proposto neste trabalho uma investigação aos modelos de classificação e segmentação de dados de imagiologia relativas a diferentes patologias recorrendo a metodologias de *deep learning* (DL).

Machine learning (ML) consiste na prática da utilização de algoritmos para extrair informação, analisar e aprender com dados. Com base nos conhecimentos adquiridos o algoritmo de ML devolve uma resposta em formato de uma determinação ou previsão. Com base em redes neuronais artificias inspiradas no cérebro biológico, as técnicas de *deep learning* (DL) utilizam redes de estruturas com múltiplas camadas denominadas de redes neuronais profundas.

1.2 Objetivos

O principal objetivo desta dissertação é o de desenvolver e implementar metodologias de *machine learning* eficientes para a classificação e segmentação de imagens médicas com vista ao reconhecimento de diferentes patologias médicas.

De forma discriminada apresentam-se de seguida alguns dos subobjetivos deste trabalho de investigação:

i. Identificar dataset de imagens médicas com características específicas para suporte ao trabalho desenvolvido;

ii. Comparar o desempenho de diferentes arquiteturas implementadas em classificação de imagens médicas;

iii. Comparar o desempenho de diferentes arquiteturas presentes na literatura;

iv. Analisar e comparar o impacto do pré-processamento de imagens;

v. Estudar e comparar diferentes arquiteturas utilizadas em segmentação de imagens;

vi. Identificar estruturas especificas a nível celular, permitindo identificar diferentes estados de desenvolvimento;

vii. Desenvolver uma plataforma web protótipo para apoio à identificação de patologias em imagens médicas de fácil acesso.

1.3 Metodologias

A primeira fase no desenvolvimento desta dissertação consistiu em uma ampla revisão de literatura sobre as temáticas de imagiologia médica, metodologias de *machine learning* (em imagens médicas) e às diferentes arquiteturas utilizadas em imagem médica. Paralelamente, a viabilidade do estudo de diferentes conjuntos de dados de imagens médicas foi investigado para suporte ao estudo. Adicionalmente, foram exploradas metodologias de classificação e segmentação com viabilidade para o estudo dos diferentes conjuntos de dados.

Após um levantamento relativo às metodologias, métodos e técnicas presentes na literatura foram desenvolvidos métodos de leitura dos dados assim como técnicas de processamento de imagens médicas. As métricas utilizadas na avaliação de desempenho dos algoritmos implementados foram investigadas, desenvolvidas e implementadas, tendo em consideração as especificidades de cada uma das tarefas propostas. Adicionalmente, foram implementadas funções para a visualização dos dados, do efeito do processamento aplicado às imagens, e dos resultados obtidos. Foram selecionadas e implementadas arquiteturas/modelos mais adequados a cada uma das tarefas, assim como a criação de funções a utilizar no processo de aprendizagem e no processo de teste.

Na última fase deste trabalho foram conduzidos diferentes testes preparatórios no sentido de calibrar e identificar os limites de simulação por forma a que fosse possível sustentar uma discussão comparativa de resultados. Após esta fase preparatória foram conduzidos todos os testes culminando com a análise crítica e comparativa dos resultados obtidos para os diferentes conjuntos de dados e modelos implementados.

1.4 Estrutura da Dissertação

Esta dissertação encontra-se organizada em cinco capítulos:

Capítulo 1: Introdução

O primeiro capítulo contextualiza o trabalho, estabelece as motivações que levaram ao desenvolvimento do mesmo, assim como, define os objetivos do trabalho. Adicionalmente, está presente uma breve descrição das metodologias utilizadas. Este capítulo apresenta, ainda, a organização da dissertação, dando uma breve descrição do que será referido em cada capítulo.

Capítulo 2: Revisão Bibliográfica

O segundo capítulo trata dos antecedentes teóricos que suportam o trabalho desenvolvido e por isso é subdividido em quatro subcapítulos. O primeiro subcapítulo é uma breve revisão focada na imagiologia médica seguido de um subcapítulo onde são exploradas patologias estudadas com recurso a imagem médica. Uma revisão sobre manipulação, exploração e processamento de dados é descrita no terceiro subcapítulo. Por fim, no quarto subcapítulo consiste numa revisão sobre *Machine* e *Deep Learning* com foco nas metodologias e nas arquiteturas utlizadas no âmbito da imagem médica.

Capítulo 3: Metodologias e Desenvolvimento

O terceiro capítulo refere-se aos procedimentos adotados para o desenvolvimento do trabalho, nomeadamente, ferramentas utilizadas, conjuntos de dados e as arquiteturas implementadas.

Capítulo 4: Resultados e Discussão

O quarto capítulo centra-se nos resultados e discussão do trabalho de investigação desenvolvido, sendo dividido em três subcapítulos, que tratam da discussão dos resultados obtidos relativamente às tarefas de classificação e segmentação de imagens médicas e um último onde é apresentada a plataforma web de demonstração de resultados.

Capítulo 5: Conclusões e Perspetivas Futuras

O quinto capítulo centra-se nas principais conclusões e no trabalho futuro.

Capítulo 2: Revisão Bibliográfica

A revisão bibliográfica para o desenvolvimento desta dissertação assenta em quatro subcapítulos, nomeadamente, imagiologia médica, exemplos de patologias estudadas com recurso a imagem médica, manipulação, exploração e processamento de dados e, ainda, subcapítulo mais extenso sobre *Machine* e *Deep Learning*.

2.1 Imagiologia Médica

A imagiologia médica é uma técnica crucial em medicina, tanto no diagnóstico precoce como na precisão do diagnóstico [1]. A interação da imagiologia médica com outras áreas científicas como, por exemplo, a inteligência artificial são cada vez mais um fator crucial para a deteção de patologias precocemente.

A imagiologia médica consiste em um conjunto de processos ou técnicas onde são criadas representações visuais das partes interiores do corpo, tais como órgãos ou tecidos, com um propósito clínico a fim de monitorizar a saúde do paciente, diagnosticar e tratar doenças ou lesões (Figura 1). Devido aos avanços tecnológicos na área de imagem médica, a imagiologia médica permite a obtenção de informação do corpo humano para diferentes aplicações clínicas. Os diferentes tipos de tecnologia fornecem diferentes informações sobre a área do corpo a ser estudada ou a ser tratada [2].

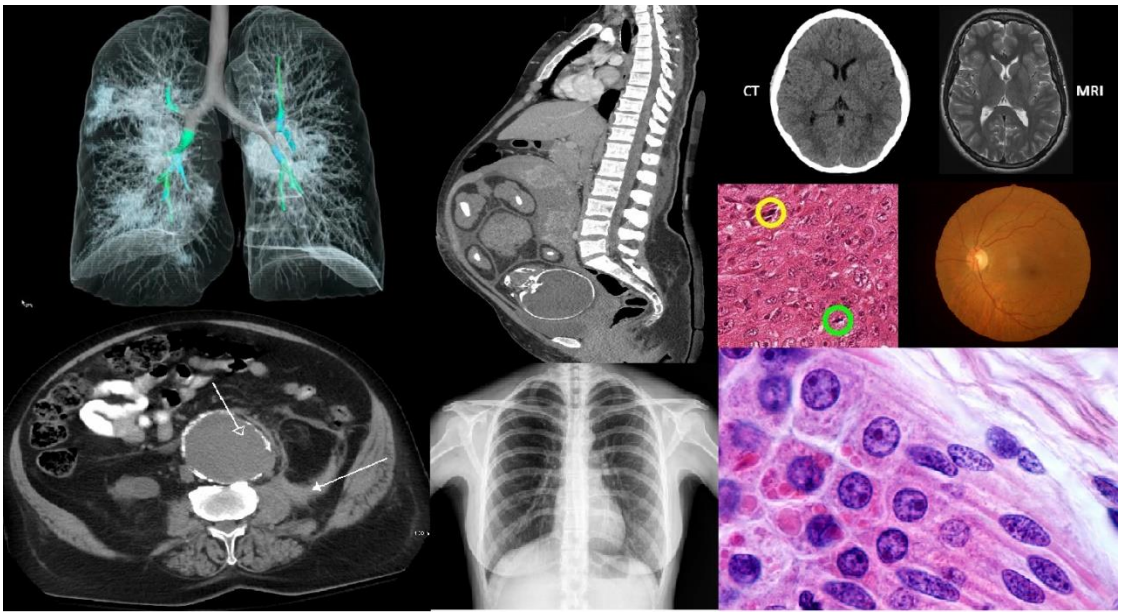


Figura 1. Exemplos de aplicações de imagem médica.

A importância da utilização de imagens médicas para serviços de diagnóstico é considerada como uma afirmação significativa da avaliação e documentação de muitas doenças. A imagiologia de alta qualidade melhora a tomada de decisões médicas e pode reduzir procedimentos desnecessários [2]. Por exemplo, as intervenções cirúrgicas podem ser evitadas se estiver disponível tecnologia de

imagem médica, como os ultrassons e as ressonâncias magnéticas. O diagnóstico de doenças como, por exemplo, o cancro, a segunda maior causa de mortalidade no mundo de acordo com a Organização Mundial da Saúde (OMS) [3], tornou-se uma tarefa mais facilitada com o uso das tecnologias de imagem médica [4].

A imagiologia médica é parte integrante do ramo de imagiologia biológica e incorpora nos seus estudos a radiologia que inclui as seguintes tecnologias:

Radiografia

A radiografia é uma das primeiras técnicas de imagiologia utilizadas na medicina moderna. Esta tecnologia utiliza um feixe de raios X para poder visualizar material não composto uniformemente. As imagens obtidas com recurso à radiografia ajudam na avaliação da presença ou ausência de doenças, danos ou objetos estranhos [5]. As técnicas de radiografia são ainda usadas frequentemente na avaliação de doenças como inflamações e infeções na região do abdómen e na identificação de pneumonias. São utilizadas duas formas de obtenção de imagens radiográficas:

- Fluoroscopia: são produzidas imagens de partes internas do corpo em tempo real recorrendo à utilização de doses constantes, mais baixas, de raio X.

- Radiografia projecional: forma mais usada como técnica de radiografia normalmente utilizada para determinar o tipo e extensão de uma fratura ou alterações patológicas, por exemplo em um pulmão. Esta técnica é também usada para visualizar áreas internas em torno do estômago e intestino podendo assim ajudar no diagnóstico de úlceras e de certos tipos de cancro do cólon.

Tomografia

A tomografia computorizada por emissão de fotão único (SPECT), utiliza raios gama para a imagiologia médica, onde um radioisótopo emissor é injetado na corrente sanguínea. A tomografia é utilizada para qualquer estudo de imagem gama que seja útil no tratamento especial de tumores, leucócitos, tiróides e ossos [6]. As tomografias de crânio são usadas na deteção de aneurismas, hemorragias e hidrocefalia, enquanto as tomografias ao tórax permitem a identificação de doenças vasculares e de tumores. No caso da tomografia axial computorizada, também conhecida por TAC ou TC, esta é uma técnica utilizada em exames às zonas abdominais, pulmonares, crânio e em diferentes patologias ortopédicas.

Ressonâncias Magnéticas (MRI)

Os scanners de MRI recorrem à utilização de ímanes, emitindo pulsos de frequências ressonantes dos átomos de hidrogénio de forma a polarizar e excitar moléculas de água em tecido humano [7]. A tecnologia MRI não envolve raios X nem radiação ionizante e é amplamente utilizada em hospitais dado ser uma escolha mais segura já que não utiliza radiação [8]. Uma desvantagem do MRI é o facto do exame ser mais demorado e de pessoas que possuam implantes não removíveis dentro do

corpo não conseguirem submeter-se a estes tipos de exames. Os MRI são utilizados na deteção de aneurismas, tumores, alterações nas articulações e lesões em órgãos internos.

Ultrasom

Esta técnica utiliza ondas sonoras de banda larga com altas frequências que são refletidas pelo tecido em graus variáveis para produzir um tipo de imagens 3D. Esta técnica é usada para imagiologia de órgãos abdominais, coração, mama, músculos, tendões, artérias e veias fornecendo menos detalhes anatómicos por comparação com as tomografias ou MRI, tendo como vantagem a possibilidade de serem estudadas as funções das estruturas em movimento e em tempo real sem emitir radiações [9].

Endoscopia

Esta técnica utiliza um endoscópio que é inserido diretamente no órgão para que este possa ser examinado. O tipo de endoscópio difere em função do local a examinar no corpo e pode ser realizado por um médico de clínica geral ou um cirurgião. A endoscopia é utilizada para examinar o sistema gastrointestinal, vias respiratórias, ouvidos e vias urinárias. Os principais riscos associados com este procedimento são infeção, perfuração, revestimento lacrimal e hemorragias [10]. As técnicas de endoscopia permitem a identificação de gastrites, úlceras, pólipos e de hérnias.

Termografia

Câmaras termográficas detetam radiações infravermelhas emitidas pelo corpo que criam imagens térmicas com base nestas mesmas radiações recebidas. A quantidade de radiação aumenta com o aumento da temperatura corporal. Desta forma, a termografia ajuda a verificar variações de temperatura ao longo do corpo do paciente, sendo também possível captar objetos em movimento em tempo real [11]. A termografia é usada na deteção de determinados tipos de tumor e de problemas circulatórios.

Medicina Nuclear

A imagem obtida com esta técnica é conseguida através da tomada de radiofármacos, onde posteriormente, os detetores gama externos captam e formam imagens das radiações que são imitidas pelos radiofármacos. Esta técnica consiste no oposto aos raios X, em que as radiações são emitidas através do corpo a partir do exterior, neste caso as radiações são emitidas a partir do interior do corpo [6]. Técnicas de medicina nuclear são usadas no estudo de funcionamento de diferentes órgãos e sistemas corporais. É ainda possível, não só detetar, como tratar alguns tipos de tumor.

Para além das tecnologias mencionadas anteriormente existem outras técnicas de imagem que ajudam e fornecem soluções para as mais variadas patologias. No entanto, estas não são consideradas como imagiologia médica direta. Técnicas como a eletroencefalografia (EEG),

magnetoencefalografia (MEG), eletrocardiograma (ECG) produzem dados de forma gráfica com relação ao tempo com informações importantes sobre o corpo humano.

Em suma, desde cedo que o recurso à imagem médica para a deteção de patologias é um campo de estudos extremamente explorado, sendo que o recurso a técnicas de análise, tratamento e exploração de imagens sempre foi de grande interesse para a comunidade médica. Contudo, toda a evolução destas técnicas tem como necessidade a intervenção humana especializada na análise das imagens. Assim, o desenvolvimento de técnicas automatizadas tem sido o foco por parte da comunidade científica.

2.2 Patologias estudadas com recurso a imagem médica e metodologias de *Deep Learning*

Retinopatia Diabética

A diabetes *Mellitus* é uma doença metabólica categorizada em dois tipos: tipo 1, que é caracterizado pela ausência de produção de insulina pelo pâncreas, e o tipo 2, em que o corpo não responde à insulina. Em ambos os casos, é originada uma elevada taxa de açúcar no sangue. A retinopatia diabética é um distúrbio ocular devido à diabetes que resulta em cegueira permanente. De acordo com a organização mundial de saúde (OMS) a diabetes é a principal causa de cegueira, insuficiência renal e ataques cardíacos, sendo que o número de pessoas que sofre de diabetes aumentou de 108 milhões em 1980 para 422 milhões de pessoas em 2014 [3].

A retinopatia diabética pode ser controlada e curada se diagnosticada numa fase precoce, através de testes de rastreio da retina. Os processos manuais para detetar estas patologias são demoradas devido à indisponibilidade de equipamento e aos conhecimentos necessários para efetuar o teste. A deteção automática da retinopatia diabética com base em *Deep Learning* (DL) tem mostrado resultados promissores. A investigação efetuada por Gulshan *et al.* (2016) [12] com recurso a redes neuronais convolucionais profundas (DCNN), utilizando os conjuntos de dados *EyePACS-1* [13] e *Messidor-2* [14], alcançou resultados de 97.5% de sensibilidade e 93.4% de especificidade assim como 96.1% de sensibilidade e 93.9% de especificidade, respetivamente [12]. A rede neuronal utilizada no trabalho referido é a arquitetura *Inception-V3* proposta por Szegedy *et al.* (2016) [15].

Com os avanços nos métodos de rastreio automático da retinopatia diabética o diagnóstico tem ocorrido num estado mais precoce da patologia. Agências reguladoras da área da saúde estão a trabalhar com empresas, como a *Google*, para avaliar as melhores tecnologias em estudos clínicos, a fim de obter procedimentos e padronização nos exames à patologia [16].

Deteção de Elementos Histológicos e Microscópicos

O estudo da Histologia, do grego *hystos* (tecido) e *logos* (estudo), diz respeito ao estudo da célula, grupo de células e de tecidos, bem como, das suas estruturas, formação e função. A tecnologia de imagem microscópica é utilizada para detetar as alterações microscópicas que ocorrem a nível

celular. A obtenção de imagem com tecnologias microscópicas inclui processos de fixação, microtomia, inclusão, secção e de coloração. Estas técnicas permitem a observação de tecidos de modo a avaliar morfologia e arquitetura, a diagnosticar diferentes patologias e identificar substâncias [17]. A deteção e determinação do número de células em estado mitótico permite avaliar a taxa de proliferação celular e, consequentemente, se estamos perante uma patologia, nomeadamente cancro, assim como o grau de agressividade da mesma. Portanto, a determinação deste indicador é de maior relevância no processo de diagnóstico [17].

Além da deteção de carcinomas, a imagem microscópica é utilizada para a deteção de outras patologias como, por exemplo, a malária e doenças causadas por parasitas intestinais. O parasita *Genus plasmodium* é a principal causa da malária, sendo que a imagem microscópica consiste no método padrão para a deteção deste. Bactérias do tipo *Mycobacteria* na expetoração são a principal causa da tuberculose. Uma vez mais, a imagem microscópica, juntamente com métodos de coloração fluorescente, são metodologias padrão no diagnóstico da tuberculose.

Um estudo conduzido pelo Departamento de *Computer Science* de Warwick, em Inglaterra, recorreu ao conjunto de dados *CRCHistoPhenotypes* [18] que envolve 100 imagem histológicas de H&E coloridas de adenocarcinomas colorretais, usou um modelo de *Deep Learning* para a deteção e classificação de cancros associados. O modelo usado consistia numa rede neuronal convolucional (CNN) com restrições espaciais para a deteção e classificação dos núcleos, potenciando a análise da morfologia de tecidos, com o objetivo de criar uma ferramenta útil na compreensão do microambiente do tumor [18].

Investigações efetuadas no âmbito da malária e da tuberculose utilizaram redes neuronais convolucionais profundas (DCNN), apresentando resultados promissores na deteção da malária em imagens plasmáticas e na deteção de bacilos de tuberculose [19]. A DCNN usada era composta por uma camada de convolução com 7 filtros (3X3), uma camada de *pooling*, uma segunda camada de convolução desta vez com 12 filtros (2X2) e por fim a camada de ligação completa (FCL) [19].

Deteção de Doenças Gastrointestinais

O sistema gastrointestinal é composto por todos os órgãos envolvidos na digestão de alimentos e na absorção dos nutrientes contidos nestes, desde a boca até ao ânus. A digestão e absorção dos nutrientes é afetada pela ocorrência de inflamações, hemorragias, infeções e cancros no sistema gastrointestinal. As atuais tecnologias de imagem desempenham um papel vital na deteção e diagnóstico das patologias relacionas com o sistema gastrointestinal. Tecnologias essas que incluem a endoscopia tradicional e endoscopia com recurso a cápsulas sem fios (WCE), enteroscopia, tomografias e ressonâncias magnéticas.

Uma vez mais, o diagnostico assistido por computador para a deteção de hemorragias gastrointestinais é uma área com grande atividade na investigação. Modelos baseados em *DCNN´s* para a deteção de hemorragias, em imagens obtidas com endoscopias com recurso a cápsulas sem fios (WCE), têm sido utilizados, apresentando valores de *F1-Score* de 0.9955, superando em muitos

casos as abordagens mais avançadas na deteção de hemorragias [20]. As endoscopias WCE constituem um exame indolor, podendo adquirir um grande número de imagens do interior do intestino. A angiectasia gastrointestinal (GIA) é a lesão vascular do intestino delgado mais comum, com o risco inerente de hemorragias. Investigações efetuadas com recurso a *DCNN´s* para a segmentação e classificação semântica foram desenvolvidas apresentando sensibilidade de 100 % e especificidade de 96% [21].

Imagem Cardíaca

As tomografias computorizadas e ressonâncias magnéticas são as tecnologias mais utilizadas para o estudo de imagem cardíaca. Algumas das tarefas realizadas no âmbito de estudo de imagem cardíaca são a deteção de depósitos de cálcio em tomografias computorizadas, medição do *lúmen* em tomografias de coerência ótica, segmentação de ventrículos em ressonâncias magnéticas e predição de doenças obstrutivas em tomografias de emissão monofotónica. Estudos efetuados no âmbito de criação de sistemas automáticos para análise cardiovascular têm sido realizados fazendo uso de modelos de *Deep Learning* [22]. Atualmente, podem ser encontradas aplicações de *Deep Learning* abrangendo quase todos os aspetos da imagem cardiovascular (Figura 2), desde a ecocardiografia até à fluoroscopia intraoperatória. Apesar da Figura 2 correlacionar uma ligação entre aplicação e algoritmo, não significa que outras abordagens não possam ser utilizadas.

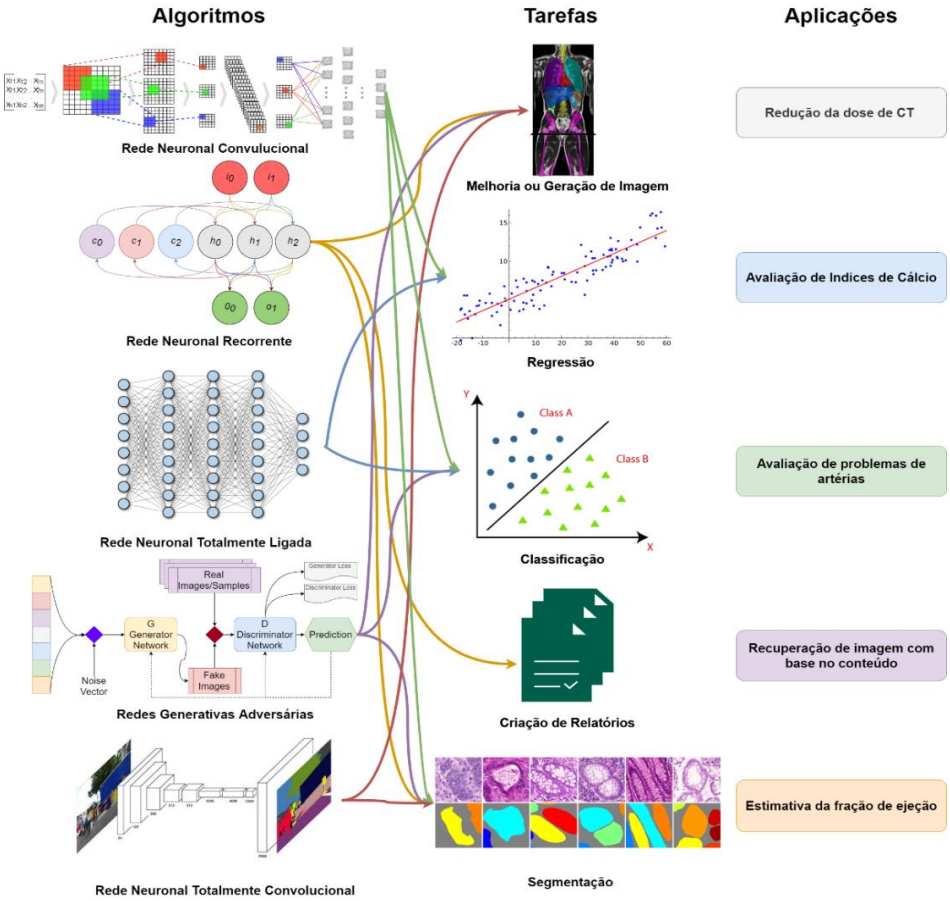


Figura 2. Exemplos de aplicações, tarefas e respetivos algoritmos usados no âmbito da imagem cardíaca. Adaptado de Litjens *et al.* (2019) [22].

Deteção de tumores

O crescimento anormal de células em qualquer parte do corpo, origina por vezes massas de tecido às quais se dá o nome de tumor ou neoplasma. As células do nosso corpo sofrem ciclos de desenvolvimento, envelhecimento, morte e finalmente, a substituição destas por novas células. Este ciclo é interrompido em caso de presença de diferentes cancros. Existem dois tipos de tumor: os benignos, não cancerígenos, e os malignos que são cancerígenos [23]. O tumor benigno não apresenta riscos elevados e normalmente não se agarra a uma parte do corpo de forma a propagar-se. No caso de tumores malignos, estes são extremamente nocivos, com grande probabilidade de propagação a outras partes do corpo. A propagação de um tumor maligno dificulta o tratamento assim como o prognostico. A título de exemplo, um dos estudos pioneiros na deteção de cancro da mama com recurso a mamografias digitais e *Machine Learning* utilizou 482 imagens mamográficas das quais 246 apresentavam tumores [24]. O conjunto de dados utilizado foi construído utilizando uma câmara de raios-X (*Senographe 2000D*) e analisado por radiologistas do Hospital da província chinesa *Liaoning.* Foram utilizados dois modelos de *Machine Learning*, sendo que um deles consistia em uma rede neuronal simples de camada única e um modelo que consistia numa SVM, máquina de suporte vetorial. As imagens foram submetidas inicialmente à passagem por uma série de técnicas de pré-processamento como as reduções de ruído, realce e segmentação de extremos e, de seguida, à extração de características geométricas e contextuais. As características extraídas foram introduzidas nos modelos e treinadas através de uma abordagem de validação cruzada [25]. Os resultados obtidos com a utilização da rede neuronal originaram 84 respostas erradas, enquanto a SVM apresentou 96 respostas erradas em 482 imagens.

Deteção de Alzheimer e Parkinson

A doença de Alzheimer diz respeito a uma perturbação cerebral irreversível que progride lentamente, destruindo a memória e as capacidades de pensamento, incapacitando o doente de realizar as tarefas mais simples. O diagnóstico exato da doença de Alzheimer desempenha um papel crucial no tratamento dos doentes, particularmente na fase inicial da doença. Quanto à doença de Parkinson, esta consiste em uma desordem neurológica que provoca o declínio progressivo do sistema motor devido à desordem dos gânglios basais presentes no cérebro. A doença apresenta-se inicialmente com tremores nas mãos, seguidos de movimentos lentos, rigidez e perdas de equilíbrio. Existem diferentes modelos baseados em redes neuronais convolucionais (CNN) no estudos e deteção das doenças de Alzheimer e de Parkinson [26-28]. A *LeNet-5*, modelo baseado nas CNN, foi usado na deteção de Alzheimer em dados oriundos de imagens de ressonância magnética funcional [26]. O modelo foi treinado em 270900 imagens e testado em 90300 imagens obtendo uma precisão média de 96.85% [26]. Um outro estudo focado na deteção da doença de Parkinson utilizou uma DCNN apresentando resultados superiores quando comparados com arquiteturas baseadas na CNN, como o caso da *LeNet-5* [27]. O conjunto de dados elaborado pela *Parkinson´s Progression Markers Initiative* foi usado, juntamente com uma DCNN, obtendo classificações em precisão média de 95.1% e AUC de 97% [27].

2.3 Manipulação, Exploração e Processamento de Dados

A implementação e uso de metodologias de *Machine Learning* (ML) e de *Deep Learning* (DL) requerem um conjunto de competências básicas de manipulação e processamento de dados, álgebra, matemática e estatística. Deste modo, este subcapítulo tem como objetivo a apresentação de tópicos pertinentes para o desenvolvimento destas metodologias, atuais desenvolvimentos e diferentes arquiteturas de metodologias com base em *machine learning* reproduzindo desta forma o atual estado da arte.

2.3.1 Manipulação e Exploração de Dados

O armazenamento e manipulação de dados é umas das primeiras etapas na implementação e teste de metodologias de ML e de DL. Existem duas etapas que se destacam na exploração dos dados: i) aquisição; e ii) processamento, depois de adquiridos e armazenados. Quanto à resolução de problemas de aquisição, extração e exploração de dados existe um procedimento geral que envolve cincos passos (Figura 3) [29, 30]:

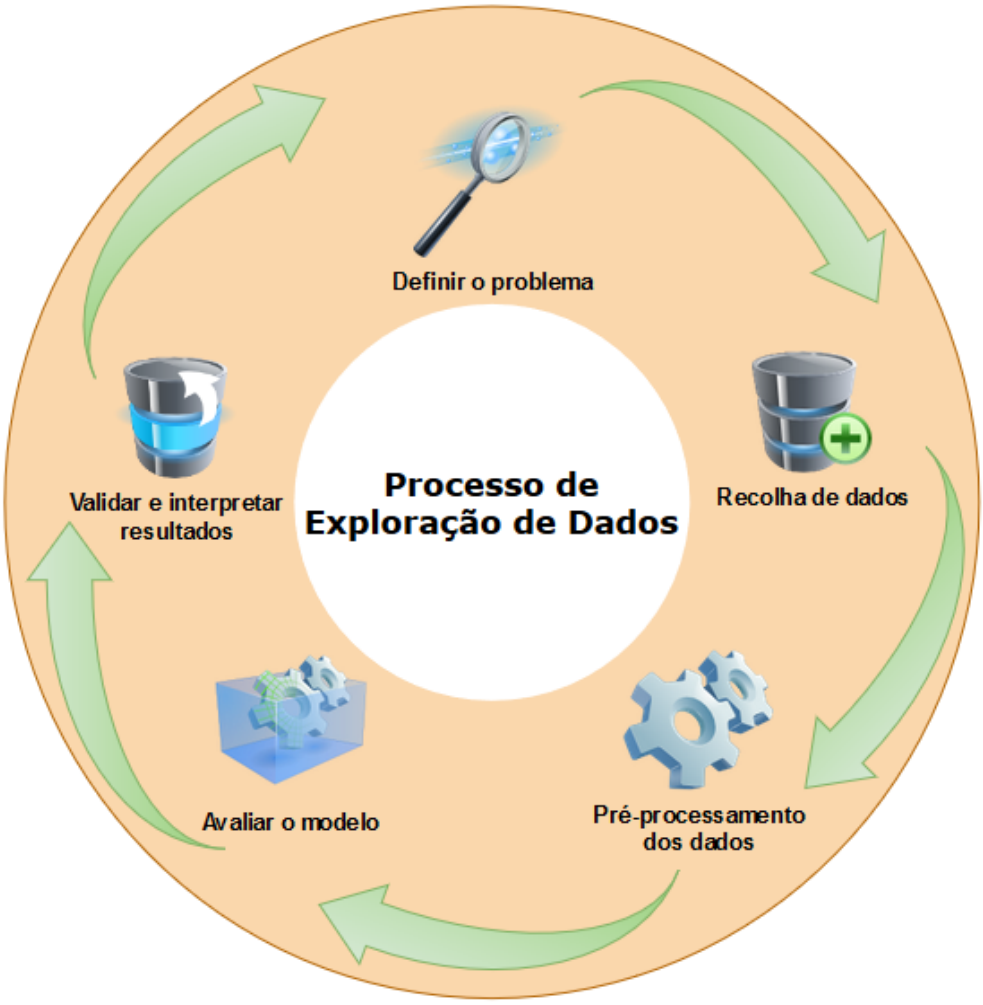


Figura 3. Processo de Exploração de Dados. Adaptado de Kantardzic *et al.* (2011) [29].

Definir o problema: Declarar o problema e formular a hipótese

A maioria dos estudos de modelação de dados são realizados num determinado domínio de aplicação. O domínio de conhecimentos específicos é imperativo para que se consiga alcançar uma boa resolução do problema. No entanto, inúmeros estudos tendem a centrar-se na exploração de dados em detrimento de uma clara declaração e formulação do problema. Aquando da declaração e formulação do problema são especificadas variáveis para as dependências desconhecidas e, se possível, uma forma geral desta dependência como hipótese inicial. É comum a existência de várias hipóteses para um único problema nesta fase inicial. É pertinente nesta fase, a perícia combinada entre o domínio na área do problema e respetiva aplicação, assim como, na exploração de dados. Apesar de pertinente na fase inicial, a cooperação entre perícia na aplicação e exploração de dados deverá continuar durante todo o processo de exploração de dados [29].

Recolha de dados

A fase de recolha de dados pressupõe conhecimento sobre como foram gerados os dados, bem como a sua aquisição. Por norma, existem duas possibilidades distintas para a recolha de dados. A primeira é conhecida como abordagem “experiência” e diz respeito ao processo de obtenção de dados a partir de um perito. A segunda possibilidade, a abordagem observacional, consiste quando o perito não pode influenciar o processo de obtenção ou aquisição de dados. Quanto à distribuição da amostragem de dados, esta é tipicamente desconhecida após a recolha de dados, ou é parcialmente desconhecida e implicitamente fornecida no processo de recolha de dados. A devida compreensão sobre o facto de o procedimento adotado na recolha de dados afetar a respetiva distribuição teórica poderá ditar uma melhor modelação do problema, da recolha de dados, assim como, da interpretação dos resultados finais. Além da preocupação associada à recolha de dados e dos procedimentos associados, é importante que os dados utilizados, para criar um modelo e os dados utilizados posteriormente para o testar, sejam provenientes da mesma fonte de dados [29, 31].

Pré-processamento dos dados

No pré-processamento de dados são duas as tarefas mais comuns [29, 32]:

a) Detetar e remover, quando necessário, dados anómalos: como parte da fase de pré-processamento deverão ser detetadas e eventualmente removidas anomalias, ou desenvolver métodos de modelação robustos que sejam insensíveis às anomalias referidas [33].
b) Dimensionamento, codificação e seleção de características: durante o pré-processamento de dados são várias as etapas a contemplar. Ao não contemplar, por exemplo, o dimensionamento dos intervalos de dados com uma característica com intervalo compreendido entre [0, 1] e um outro entre [-100, 1000], estes sendo usados com a mesma técnica, não só o peso tido em consideração não será o mesmo, como os dados finais terão sido influenciados pela diferença entre os intervalos. Por conseguinte, recomenda-se

escalar os intervalos uniformemente, e trazer neste caso ambas as características com o mesmo peso para uma análise mais aprofundada [32].

Avaliar o modelo: Implementar a técnica apropriada para a exploração dos dados

A seleção e implementação da técnica apropriada de exploração de dados é a principal tarefa nesta fase. Este processo não é simples, e na prática, a implementação passa pelo teste com vários modelos, e a seleção do melhor é uma tarefa adicional [29, 34].

Validar e interpretar os resultados e tirar conclusões

O pretendido e obtido na maioria dos casos de uso de modelos de exploração de dados é uma ajuda à tomada de decisão. Para isso, tantos os modelos usados como os resultados obtidos precisam de ser facilmente interpretáveis para serem úteis. Caso contrário, as decisões dificilmente serão tomadas com base na resposta de um modelo extremamente complexo, ao ponto de não ser possível explicar os seus resultados. Associado à interpretação, o utilizador final não quererá um resultado complexo, sendo-o, o utilizador não o conseguirá compreender, resumir e muito menos interpretar e utilizar para uma tomada de decisão bem-sucedida [29, 30].

A extração de características pode ser vista como o processo de extração de conhecimento e descoberta de importantes padrões a partir de dados [35]. Os dados estão a tornar-se cada vez mais abundantes sendo que, atualmente, devido à facilidade de adquirir mais armazenamento e mais poder computacional, armazenamos até à ínfima informação que de outra forma deitaríamos fora há alguns anos atrás. Consequentemente, à medida que a abundância de dados cresce e as máquinas que podem realizar este tipo de tarefas se tornam comuns, também se torna necessário compreender o significado desses dados, uma vez que informações pertinentes podem ser extraídas dos mesmos. Essa mesma informação é o que atraiu pessoas e empresas a utilizar a prospeção de dados, a pesquisa de algo novo e não trivial em grandes quantidades de dados.

A maioria das abordagens de exploração e extração de dados existentes procuram padrões em tabelas de dados individuais, denominadas por metodologias de exploração proposicional. Embora sejam úteis para conjuntos de dados não muito complexos, no caso de conjuntos de dados que tenham tabelas com múltiplas relações, estes resultam em perda de significado ou de informação. *Relacional Data Mining* (RDM), frequentemente referido como *Multi Relational Data Mining* (MRDM) é uma abordagem que tem como objetivo resolver esse problema. O RDM procura padrões que envolvam múltiplas relações de tabelas, a fim de lidar com os dados complexos do mundo real [36]. Estas abordagens têm sido aplicadas a uma variedade de áreas, notavelmente na área da bioinformática [37].

Dois dos principais objetivos da exploração de dados dizem respeito à previsão e descrição [29, 38]. A previsão é quando o algoritmo de aprendizagem utiliza os dados atuais para fazer previsões futuras enquanto que a descrição tenta caracterizar as propriedades dos dados, num respetivo conjunto de dados [35].

De forma a alcançar o objetivo de extrair informação útil a partir de dados, existem inúmeras metodologias que podem ser usadas, tanto na exploração de dados, como, consequentemente, na utilização desses mesmos dados em aplicações de *Machine Learning* assim como de *Deep Learning.* Serão brevemente descritas algumas das terminologias e metodologias usadas:

Classificação

Grupo de algoritmos com o propósito de classificar os dados em um número finito de etiquetas/anotações, valores de classe, aprendendo uma função que mapeia e caracteriza os objetos com a correspondente etiqueta.

Regressão

Algoritmos com o propósito de tentar construir uma função de previsão de aprendizagem, que mapeia um determinado elemento para o valor real previsto pela função.

Sumarização

Consistem na tarefa descritiva de tentar representar os dados de uma forma concisa, mantendo ao mesmo tempo as principais características do conjunto de dados.

***Clustering* (Agrupamento)**

Grupos de algoritmos com a tarefa descritiva de identificar um conjunto de *clusters* ou categorias para definir dados semelhantes. Um subcampo do *Clustering* denominado por *Clustering* conceptual, visa não só agrupar os dados, mas também descobrir o significado por trás de cada *cluster.*

Deteção de anomalias

Algoritmos com o propósito de encontrar valores invulgares, ou seja, valores que se desviam do que é normal no conjunto de dados. Estes valores são considerados anormais e podem ser interpretados como erros ou ao mesmo tempo como valores interessantes, que devem ser investigados mais aprofundadamente.

Modelos de avaliação

Grupo de algoritmos de avaliação tentam encontrar dependências entre variáveis ou valores de uma característica, em um conjunto de dados.

Dois dos maiores desafios encontrados em manipulação e exploração de dados, dizem respeito aos conjuntos de dados em desequilíbrio e à dimensionalidade reduzida.

O processo de redução de dimensão consiste na descoberta de representações compactas de alta dimensão e definição dos dados [39]. Em casos onde os dados possuem dimensões elevadas, é pertinente o uso de metodologias de extração de características, para que seja possível extrair e

analisar a sua representação compacta, em comparação com os dados em bruto. Algumas das práticas utilizadas neste processo estão brevemente descritas de seguida.

Análise de componentes principais (PCA)

PCA consiste em uma técnica linear cujo objetivo é codificar dados de alta dimensão numa representação dimensional inferior. Encontrar a informação mais importante dentro dos dados e gerar um conjunto de variáveis ortogonais chamadas componentes principais, para melhor diferenciar os pontos de dados [40]. Em suma, o princípio por detrás deste método é a maximização da variação da transformação ortogonal dos dados em bruto.

***Kernel* PCA (KPCA)**

É uma extensão do PCA, mas ao contrário da metodologia usada no PCA, a KPCA tenta encontrar uma espaço não-linear de baixa dimensão [41]. São usados métodos de *kernel* para calcular os principais componentes em espaços de características de alta dimensão.

Autoencoders

Representam um tipo de rede neuronal artificial que é normalmente utilizada em aprendizagem. Estes modelos tentam codificar os dados e reconstruí-los ao mesmo tempo que tentam minimizar o erro, encontrando representações compactas dos dados, mas mantendo a informação. Uma descrição mais pormenorizada será apresentada de seguida no subcapítulo 2.4.6.1.

Conjuntos de dados em desequilíbrio

Uma questão que está frequentemente presente nos dados atuais são os conjuntos de dados em desequilíbrio. A falta de amostras de uma classe é um exemplo claro desta problemática. Quando confrontados com um conjunto de dados desequilibrado, os classificadores serão frequentemente enviesados para a classe maioritária, falhando em realmente aprender a diferença entre a classe maioritária e a classe minoritária [42].

Aprendizagem Sensível a Custos

Muitas vezes denominada por aprendizagem automática sensível ao custo, é normalmente útil quando os dados são conhecidos e é sabido de antemão que as amostras da classe minoritária podem receber mais importância, quando comparado com as classes de amostras com mais frequência. Com recurso a esta técnica, em vez de cada instância ser classificada como correta ou incorreta, cada classe recebe um custo por classificação de errado. Assim, em vez de tentar otimizar a precisão, o problema é então tentar minimizar o custo total da classificação errónea [43]. Da mesma forma que nem todos os erros de classificação são iguais, a maioria dos algoritmos de aprendizagem também assume que os erros de previsão, efetuados por um classificador, são os mesmos levando também esta suposição a classificações incorretas. A maioria dos classificadores assume que os custos por classificação errada (falsos positivos e falsos negativos) são os mesmos, quando na maioria das aplicações do mundo real, esta suposição não é de todo verdadeira [44]. A

gravidade do estudo com base em um conjunto de dados em desequilíbrio, sem que nada seja efetuado de forma a colmatar estas discrepâncias, pode ser esclarecida com o seguinte exemplo: em diagnósticos médicos de um determinado cancro, se um cancro for considerado como a classe positiva, e o não cancro (individuo saudável) como negativo, então a classificação errada de um paciente como não tendo cancro, ou seja, o doente é na realidade positivo mas é classificado como negativo (falso negativo), é muito mais grave do que o erro de um falso positivo [45]. A aprendizagem sensível ao custo é um tópico de investigação relativamente novo no conjunto de estudos da Inteligência Artificial. Cada vez mais os modelos de aprendizagem automática necessitam de ter sensibilidade ao custo associado às interações com que são sujeitos e, na melhor das hipóteses, ter o custo em conta no processo de adaptação do modelo [46]. Em suma, a utilização de aprendizagem sensível aos custos é um tipo de aprendizagem que têm em consideração os custos de uma classificação errada. O objetivo destas técnicas é minimizar ao máximo o custo total das classificações [45].

Amostragem de Dados

É uma forma de modificar o conjunto de dados com recurso a alguns mecanismos, a fim de o equilibrar. A reamostragem pode ser feita de várias formas, quer por subamostragem, amostragem excessiva ou métodos híbridos que combinam ambas as estratégias [47]. Os métodos mais comuns e básicos envolvem a amostragem excessiva e subamostragem aleatórias, onde as amostras são duplicadas e removidas, respetivamente. No entanto, estas estratégias provaram não ser as melhores uma vez que não possuem heurísticas, como tal, as desvantagens chegam com o problema mencionado, a amostragem excessiva aleatória leva frequentemente a ajustes excessivos (*overfitting*), enquanto a subamostragem aleatória pode levar à eliminação de amostras importantes [42]. Atualmente são utilizadas versões melhoradas das metodologias referidas, um desses exemplos é o SMOTE, técnica de amostragem excessiva de minoria sintética [48]. Com esta técnica, a classe minoritária é sujeita a inúmeros ajustes com base nas semelhanças de espaço entre as suas amostras. Para criar uma amostra sintética são considerados os vizinhos *K-Nearest* (*K-Nearest Neighbors*) e selecionados, aleatoriamente, um para multiplicar a sua diferença vetorial de característica, por um número aleatório entre 0 e 1, e adiciona este vetor ao vizinho selecionado. No entanto, esta abordagem tem alguns inconvenientes, pois quando está a gerar amostras sintéticas não considera exemplos vizinhos, o que pode levar a uma generalização excessiva e aumenta a sobreposição entre classes [42, 47] . A fim de lidar com esta desvantagem, métodos de amostragem adaptativos, tais como Amostragem Sintética Adaptativa (ADASYN), foram propostas [49, 50]. ADASYN utiliza um método para criar dados de acordo com as suas distribuições, ou seja, são geradas mais amostras para a classe minoritária, amostras essas que são mais difíceis de aprender em comparação com amostras que são mais fáceis de aprender [49]. Para além do ADASYN, existem também outras formas de lidar com as questões do SMOTE [48], uma dessas formas é utilização de técnicas de limpeza de dados como SMOTE + Tomek ou SMOTE + ENN [51] que limpam as sobreposições indesejadas entre classes após a amostragem excessiva, cada uma usando métodos diferentes para a tarefa.

2.3.2 Processamento de imagem

As imagens em bruto contêm ruído, pelo que o primeiro passo no processamento destas é o pré-processamento, ou seja, melhorar a qualidade da imagem a ser utilizada, realçar as características pertinentes nestas e remover as informações indesejadas como os ruídos referidos. Podem ocorrer diferentes imprecisões na classificação, se o processamento não for devidamente acautelado. Diferentes filtros podem ser aplicados para a remoção de dados indesejados, como por exemplo o ruído *gaussiano* e o de *poisson*. Filtro médio e mediano, filtro mediano adaptativo e o filtro *gaussian* são alguns exemplos dos métodos de filtragem básicos mais utilizados. Os ruídos da imagem devem ser removidos ou ajustados através de tarefas de pré-processamento de imagem com metodologias de ajuste de contraste, remoção de efeitos e correção de cor, suavização de imagem e normalização. A combinação certa das tarefas de pré-processamento proporciona uma maior precisão nas classificações obtidas com as imagens usadas.

As imagens de ressonâncias magnéticas (MRI), por exemplo, são inicialmente convertidas em escalas de cinzentos e depois sujeitas a ajustes de contraste usando operações de suavização [52]. No caso de imagens tomográficas (TAC) obtidas para o diagnóstico de cancro de pulmão, estas são pré-processadas, primeiro convertendo-as em escalas de cinzento, seguindo-se a aplicação de métodos de normalização e redução de ruídos. Posteriormente estas imagens são convertidas em imagens binárias para que sejam removidas as partes indesejadas [53]. O pré-processamento em imagens oriundas de estudos de cancro da mama consiste particularmente na delineação do tumor, extração da borda peitoral e a supressão do músculo peitoral. O pré-processamento de imagens de cancro da mama é particularmente pertinente, dada a quantidade de ruído existente nestas imagens [54]. Em diagnósticos de cancro da próstata, são obtidas imagens de ultrassom, que possuem ruído inerente dos equipamentos de aquisição e uma resolução de imagem baixa. Neste caso, as metodologias de pré-processamento utilizadas para a supressão de ruídos consiste na utilização de filtragem não linear com estrutura em árvores (TSF), transformadores de ondas direcionais e transformadores de onda de árvore estruturada (TSWT) [55].

Depois do pré-processamento da imagem completo, aguarda-se o pós-processamento onde a tarefa é extrair as características pertinentes. De forma a garantir a aquisição das características, no estudo de caso da segmentação para imagens usadas no diagnostico do cancro, os métodos mais comuns são a analise de componentes principais (PCA), transformações *wavelet* por pacotes (WPT) [56, 57], matriz de ocorrências de níveis de cinzentos (GLCM) [58], espectro de potência de *Fourier* (FPS) [59], *kernel's* derivados de *gauss* [60] e características de fronteiras de decisão [61].

2.4 *Machine* e *Deep Learning*

Em aprendizagem automática, *Machine Learning* (ML), são desenvolvidas e estudadas metodologias que dão aos computadores a capacidade de resolver problemas através de um processo de aprendizagem que é alimentado de "experiências". O principal objetivo da aprendizagem automática é o de providenciar modelos que consigam captar ou modelar determinadas relações a partir de dados, ou por outras palavras, aprender a partir da experiência e que posteriormente consiga reproduzir esses mesmos comportamentos [62]. As experiências passadas aos modelos de ML são fornecidas sob a forma de dados, que serão usados durante o processo de aprendizagem. Através de processos de otimização, os modelos são ajustados de forma a produzir previsões precisas com base no conjunto de dados de treino, para que depois deste processo, este seja capaz de generalizar, e possa assim fornecer previsões precisas em dados nunca vistos. A capacidade de um modelo de generalizar é tipicamente estimada durante o processo de treino, utilizando um conjunto de dados separado denominado por conjunto de dados de validação, que o usa como *feedback* para uma melhor afinação e parametrização do modelo. No final de várias iterações de treino e afinação, o modelo final é avaliado em um outro conjunto de dados separado denominado por conjunto de dados de teste, de forma a analisar o desempenho do modelo quando exposto a novos dados [62].

Existem diferentes tipos de aprendizagem automática, normalmente categorizados de acordo com a forma como os modelos utilizam os seus dados de entrada para o processo de aprendizagem. Em *Reinforcement Learning* (RL) são construídos agentes que aprendem com os seus respetivos ambientes através de tentativa e erro, ao mesmo tempo que se otimizam recorrendo a funções especificas. Em modelos de aprendizagem não supervisionada, os modelos são desenvolvidos para descobrirem padrões nos dados sem que exista orientação, ou seja, sem que os dados de treino possuam uma etiqueta que caracterize determinada amostra. Os modelos baseados em agrupamento (*clustering*) são um excelente exemplo desta tipologia. No entanto, os modelos de ML amplamente utilizados atualmente fazem parte do grupo da aprendizagem supervisionada. Os modelos deste grupo recebem um conjunto de dados anotados com etiquetas e são desenvolvidos para que após o processo de treino, com base na informação descoberta, estes possam produzir classificações corretas em conjuntos de dados novos e nunca vistos.

Na aprendizagem supervisionada é oferecida uma ajuda ao computador/algoritmo, onde esta ajuda, que na verdade é o resultado de uma determinada instância de um conjunto de dados, é fornecida juntamente com os dados de entrada. O objetivo deste tipo de aprendizagem é generalizar, ou seja, o modelo precisa de descobrir como obter os resultados corretos para os novos conjuntos de dados que não estavam presentes no conjunto de treino. Um problema comum a ter em conta com a utilização desta técnica são os excessivos ajustamentos, também conhecido como *overfitting,* perdendo assim a capacidade de generalizar. Esta problemática acontece quando o modelo decora os dados em vez de determinar uma função que caracterize os dados. Nestas circunstâncias o modelo, quando confrontado com novos dados, que não estavam presentes na aprendizagem,

apresenta um mau desempenho. Também um conjunto de dados pequeno, pouco diversificado e homogéneo pode originar facilmente o *overfitting*.

Quanto aos modelos de aprendizagem não supervisionada, ao contrário da supervisionada, não lhe é apresentada uma ajuda no momento de aprendizagem, ou seja, o resultado desejado para determinada instância de um conjunto de dados não lhe é apresentado. A utilização deste tipo de modelos e técnicas é utilizado para encontrar a estrutura ou a relação entre os dados fornecidos, como, por exemplo, o agrupamento (*clustering*) que é um dos métodos de aprendizagem não supervisionada mais usados. Poderá ser uma opção viável quando se torna difícil obter classificações, ou etiquetas, para todas as amostras por exemplo quando a etiqueta é de difícil acesso aquando da recolha de informação.

2.4.1 Medidas de Desempenho

A avaliação do desempenho de um modelo é uma das etapas centrais de um processo de exploração de dados e aprendizagem automática. O nível de desempenho indica o sucesso das previsões de um conjunto de dados treinado por um modelo. Este desempenho pode ser avaliado por quatro valores diferentes [63]:

- Número de previsões corretas, ou seja, verdadeiros positivos (TP),

- Pelo número de previsões corretas, mas que não pertencem à classe correta, ou seja, verdadeiros negativos (TN),

- O número de exemplos que foram incorretamente atribuídos a uma classe, os falsos positivos (FP),

- E o número de exemplos que não foram reconhecidos como sendo de uma classe, os falsos negativos (FN).

Os referidos valores formam uma matriz, conhecida por matriz de confusão. Na Figura 4 está presente um exemplo de uma destas matrizes, resultado de uma classificação binária.

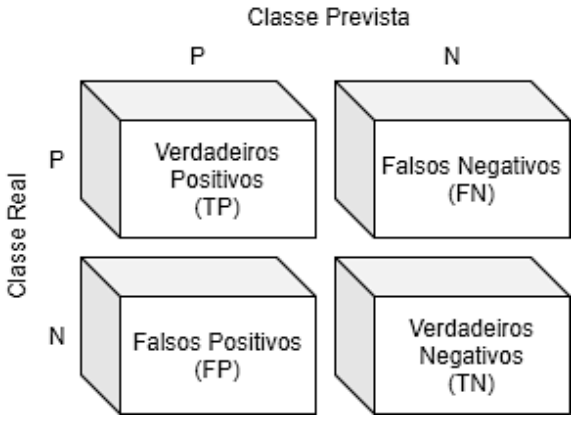


Figura 4. Exemplo de uma matriz de confusão para classificação binária.

Algumas das métricas utilizadas com recurso aos valores da matriz de confusão são a exatidão, precisão, *recall* e a pontuação F1 (P = Positivo; N = Negativo; TP = Verdadeiros positivos; TN = Verdadeiros negativos; FN = Falsos negativos; FP = Falsos positivos).

Exatidão (ACC)

A exatidão consiste na eficácia global de um modelo classificador [63]. Este cálculo mede quantas instâncias são completamente classificadas como corretas e é conseguido com recurso à seguinte equação: $ACC = \frac{TP+TN}{TP+TN+FP+FN}$.

Área sob a curva ROC (AUC)

A métrica AUC é um valor numérico que mede toda a área bidimensional por de baixo de toda a curva ROC. Esta fornece uma medida agregada do desempenho em todos os limiares de classificação possível. O AUC pode ser expresso por: $AUC = \frac{\sum R_i(I_p) - I_p(I_p+1)/2}{I_P+I_n}$, onde I_p representa o número de exemplos positivos de uma classe, I_n o número de exemplos negativos e R_i é a classificação [64].

Precisão

A precisão (*Precision*) é a proporção de classes que foram corretamente identificadas pelo modelo classificador [63]. A precisão é expressa na seguinte equação: $Precision = \frac{TP}{TP+FP}$.

Sensibilidade (Sn)

A sensibilidade também conhecida por *recall*, revela apenas quantos dos casos positivos globais são corretamente classificados corretamente e é matematicamente expressa por: $Sn = \frac{TP}{TP+FN}$ [63].

Especificidade (Sp)

Esta métrica demonstra quão exatas são as previsões negativas globais e retrata o quanto estão corretas as classificações [63]. A métrica da especificidade é expressa através de: $Sp = \frac{TN}{TN+FP}$.

Pontuação F1

A pontuação F1 (*F1 Score*) é um meio harmónico tanto de precisão como de recordação [64]. É uma métrica importante a utilizar em situações de avaliação de modelos de classificadores que fizeram uso de conjuntos de dados desequilibrados e é expressa da seguinte forma: $F1\ Score = \frac{2.TP}{2.TP+FP+FN}$.

Interseção sobre a União

A interseção sobre a união (IoU), também conhecida como índice de *Jaccard*, representa a métrica de avaliação mais utilizadas em tarefas de segmentação, deteção de objetos e de localização [65]. A deteção de objetos subdivide-se em duas tarefas: a localização, que determina a localização de

determinado objeto em uma imagem, e a classificação, que atribui a este mesmo objeto detetado uma classificação. Matematicamente, a IoU pode ser calcula da seguinte forma: $IoU\ (A,B) = \frac{|A\ \cap B|}{|A\ \cup B|} = \frac{|I|}{|U|}$, onde A e B representam respetivamente as áreas (*bounding boxes*) oriundas da previsão e as áreas de delimitação verdadeira (Figura 5). Ou seja, I representa a interseção das áreas e U representa a união destas [66].

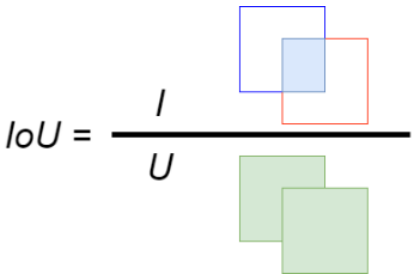


Figura 5. Interseção sobre a união (IoU).

Esta medida de desempenho desempenha um papel pertinente em aplicações de segmentação, dada a propriedade desta ser invariante a escalas. Isto significa que a largura, altura e a localização das áreas das caixas delimitadoras em consideração são tidas em conta [65].

2.4.2 Funções de Cálculo de Perdas

Entropia Cruzada Binária

A entropia cruzada binária, *Binary Cross-Entropy*, é definida como a medida da diferença entre duas distribuições de probabilidade para uma dada variável aleatória ou conjunto de eventos. Esta é expressa matematicamente por $L_{BCE}(y, y') = -(ylog(y') + (1-y)\log(1-y'))$, onde y' representa o valor previsto pelo modelo [67]. A entropia cruzada binária é amplamente usada em tarefas de classificação e segmentação (classificação de níveis de *pixéis*) [68].

Entropia Cruzada

A entropia cruzada (CE) mede o desempenho de um modelo de classificação cujo resultado está compreendido entre zero e um. A entropia cruzada aumenta na medida que a probabilidade prevista diverge da anotação ou etiqueta real. A entropia cruzada é expressa da seguinte forma: $CE = -\sum_i^c t_i \log(s_i)$ onde t_i e s_i são o valor real e a pontuação, respetivamente, do modelo para cada classe i em c [69].

Erro Médio Quadrático

O erro médio quadrático (MSE), também conhecido por perda L2, é normalmente utilizada no cálculo de perdas em aplicações de regressão (a maioria das variáveis pode ser modelada numa distribuição gaussiana). O MSE consiste na média das diferenças quadráticas entre o valor real e o valor previsto. É matematicamente expressa da seguinte forma: $MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y_i})^2$ [69].

Coeficiente de DICE

O coeficiente de DICE é amplamente utilizado em aplicações de segmentação. Este consiste em duas vezes a área de sobreposição dividida pelo número de *pixels* em ambas as imagens usadas na comparação. O DICE pode ser expresso da seguinte forma: $DICE(A,B) = \frac{2\,|A \cap B|}{|A|+|B|}$ [70].

2.4.3 Métodos/Algoritmos de Machine Learning

Decision Trees

As árvores de decisão (DT) são usadas como modelo de classificação com o objetivo de prever o valor de uma variável quando alimentada com várias variáveis de entrada. As DT são consideradas como uma abordagem comum em aplicações de classificação de dados, onde estas representam os dados em formato de árvore, as várias características são expressas pelos nós internos e as etiquetas das amostras são expressas pelos nós das "folhas".

Em aplicações de classificação, a variante das DT mais utilizada denomina-se por C4.5 [71]. No entanto, a variante relatada por Ruggieri *et al.* (2002) denominada por EC4.5, mantendo a mesma arquitetura, apresenta resultados de eficiência cinco vezes superiores quando comparado com a inicial C4.5.

Naïve Bayes

Naïve Bayes é um algoritmo de classificação baseado, tal como o nome indica, no teorema de Bayes. O teorema de Bayes pode ser analisado na seguinte equação: $P(A|B) = \frac{P(B|A).P(A)}{P(B)}$, e baseia-se no princípio de que cada característica que está a ser classificada é independente do valor de qualquer outra característica, sendo essa a razão por detrás do nome *naïve,* ingénuo. São bastante simples de implementar e de processamento rápido, sendo estas as razões principais da sua utilização. Ao atribuir um conjunto de características sendo estas probabilidades, este modelo pode prever qual a classe que tem a maior robustez para o novo dado de entrada (*input*).

Naïve Bayesian Network

As redes de *Naïve Bayesian* (NB) constituem um gráfico acíclico dirigido que descreve uma relação entre um conjunto alargado de variáveis, características, usando probabilidades. Os nós do gráfico representam as variáveis enquanto os arcos representam as relações de dependência ou independência existentes entre as variáveis (arcos diretos representam influencia entre variáveis) [72]. É importante realçar que os classificadores NB são utilizados em aplicações para realizar projeções de probabilidades e não previsões [73].

K-Nearest Neighbor

Com a utilização do *K-Nearest Neighbor* (KNN), duas amostras de dados são comparadas entre si com recurso a medidas de distância. A medida de distância pode também ser usada para minimizar

a distância entre duas amostras de dados idênticas ou para aumentar a distância entre duas amostras separadas [74].

A distância Euclidiana, também conhecida por método *k* mais próximo, é normalmente a metodologia padrão para medir a distância entre duas amostras, no entanto, esta distância pode também ser calculada com recurso aos métodos de *Hamming*, *Manhattan* ou *Markowski* [75]. A distância Euclidiana entre os pontos $[x, y]$ é conseguida por $D(x, y) = \sqrt{\sum_{i=1}^{n} |x_i, y_i|^2}$.

Support Vector Machines

As máquinas de suporte vetorial (SVM) são usadas como algoritmo classificador quando é possível definir um separador para as classes com um hiperplano e em aplicações de regressão. As SVM foram originalmente introduzidas por Vapnik *et al.* (1999) na última década do seculo XX [76]. As SVM são utilizadas em múltiplas aplicações, como por exemplo em biometria [77] e bioinformática [78]. O conceito principal das SVM consiste no agrupamento de dados e na utilização de hiperplanos para distinguir os vários grupos. Quando os dados podem ser divididos de forma o mais linear possível, este atinge níveis de precisão elevados. Quando os dados não podem ser divididos de forma linear, a utilização de mapeamento dos dados para um espaço diferente com alta dimensão pode ser conseguida usando funções de *kernel*. A seleção da função de *kernel* correta bem como os seus parâmetros são dois dos problemas a ter em conta quando as SVM são utilizadas [79].

Algumas das vantagens da utilização de SVM incluem: bom funcionamento com altos níveis de precisão quando existe uma clara separação, dos dados; eficaz em espaços de alta dimensão; bom desempenho em casos em que o número de dimensões é maior do que o número de amostras; eficiente em termos de memória dado que utiliza um subconjunto de dados de treino na função de decisão denominado por vetores de apoio. As desvantagens das SVM incluem: mau funcionamento com grandes conjuntos de dados dado o tempo elevado no processo de treino; resposta das SVM não fornece diretamente estimativas de probabilidade, estas são calculadas utilizando métodos externos; mau desempenho quando o conjunto de dados inclui muito ruído [80].

Redes Neuronais

O estudo de redes neuronais bem como de *deep learning* (DL) deriva da tentativa de criar um sistema computorizado que simule o cérebro humano [81]. Uma rede neuronal biológica é considerada como sendo o sistema mais complexo e organizado, com a capacidade de processar informação dos diferentes sentidos, tais como a visão, audição, tato, paladar e olfato, de forma eficiente e inteligente. Uma rede neuronal biológica consiste em uma série de neurónios interligados, que comunicam entre si através de um mecanismo chave para o processamento de informação denominado por sinapses. Em aprendizagem automática, *machine learning*, as redes neuronais artificiais representam uma família de modelos que imitam a elegância estrutural do sistema neuronal presente nas redes neuronais biológicas, de forma que estas aprendam padrões inerentes às observações.

Warren McCulloch, um neurofisiologista, e Walter Pitts, um matemático [82], desenvolveram uma rede neuronal primitiva com base no que era conhecido como a estrutura biológica no início da década de 1940. Em 1949, o livro intitulado "*The Organization of Behavior*" [83] foi o primeiro estudo a descrever o processo de atualização de pesos sinápticos que é hoje referida como a "*Regra de aprendizagem de Hebbian*". Em 1958, Frank Rosenblatt's [84] define a estrutura da rede neuronal denominada por *perceptron* com a tarefa de resolver problemas de classificação binária. Widrow [85], em 1962, introduz o dispositivo denominado por neurónio linear adaptativo (ADALINE), implementando o seu design em hardware. As limitações dos *perceptrons* foram destacadas por Minsji e Papert (1969) [86]. O conceito de retro propagação (*backward propagation*) de erros para efeitos de treinos do modelo é discutido por Werbose em 1974 [87]. Em 1979, Fukushima [88] propõe as redes neuronais artificiais denominadas por *Neocognitron,* com múltiplas camadas de *polling e* convolução oferecendo um mecanismo de reconhecimento de padrões visuais. A *Recurrent Neural Network* introduzida por Jordan *et al.* (1986) foram apresentadas como uma arquitetura para aprendizagem supervisionada [89]. Em 1989, Yann LeCun [90] combina as *convolution neural networks* (CNN) com a retro propagação (*backpropagation*) de forma a realizar eficazmente o reconhecimento automático de dígitos escritos à mão. Um dos mais importantes avanços em DL ocorre em 2006, quando *Hinton et al.* [91] implementa a *Deep Belief Network*, com várias camadas constituídas por máquinas de *Boltzmann* restritas, avidamente ensinando uma camada de cada vez de forma não supervisionada.

Na Figura 6 estão presentes alguns marcos históricos que definem o caminho do estudo de redes neuronais até às redes de *Deep Learning.*

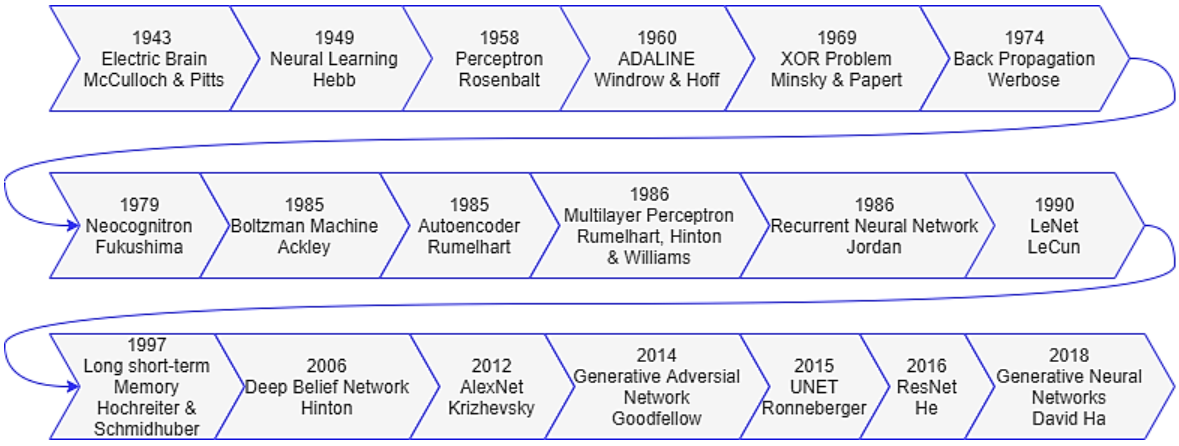


Figura 6. Desenvolvimentos significativos na história das redes neuronais. Adaptado de Macukow *et al.* (2016) e Cios *et al.* (2018) [92, 93].

2.4.4 Redes Neuronais Artificiais

As redes neuronais artificiais (ANN) formam a base estrutural de grande parte dos modelos de *deep learning.* A estrutura das ANN possui unidades simples de processamento denominadas por neurónios que se encontram interligados por ligações com um peso associado. A função descritiva do neurónio pode ser representada matematicamente por: $y = f(wx^T + b)$. Sendo $x \in \mathbb{R}^d$ o vetor

de entradas (características), $w \in \mathbb{R}^d$ o vetor de pesos, $b \in \mathbb{R}$ o *bias*, f a função de ativação (exemplo da função *Sigmoid*) e y um escalar, neste caso a saída do neurónio (nó).

De forma genérica, a construção de uma rede neuronal necessita dos seguintes parâmetros:

i) O padrão de ligações entre os neurónios (define a arquitetura),
ii) O método de determinação dos pesos da ligação (treino ou aprendizagem) e
iii) A função de ativação.

A arquitetura entre os neurónios na rede neuronal é classificada normalmente em dois tipos, rede neuronal *Feed-Forward* (FFNN) e a rede neuronal recorrente (RNN) [94]. As FFNN possuem uma ou mais camadas, onde cada uma destas compreende um ou mais neurónios (nós), e são normalmente utilizadas em aprendizagem supervisionada [95]. As RNN possuem ligações entre os nós que formam um gráfico dirigido ao longo de uma sequência temporal. Da família de arquiteturas *feedforward*, as RNN podem usar a sua memória interna para processar sequencias de entrada com comprimentos variáveis [96]. Estas características tornam as RNN aplicáveis em tarefas de reconhecimento de caracteres [97] e em aplicações de reconhecimento de voz [97].

2.4.4.1 Perceptron

O modelo neuronal artificial mais simples, tanto do ponto de vista de aprendizagem como da própria arquitetura, denomina-se por *perceptron* [98]. O modelo *perceptron* é um algoritmo utilizado normalmente para aprendizagem supervisionada em classificadores binários. Os classificadores binários determinam quando uma determinada entrada, geralmente representado por uma série de vetores, pertence a uma determinada classe específica. Em suma, um *perceptron* é uma rede neuronal de camada (*layer*) única. Estruturalmente, a arquitetura do modelo é constituída por cinco partes essenciais, valores de entrada (*inputs*) (X), pesos (W), *bias*, soma da rede e a função de ativação *f(.)* (Figura 7).

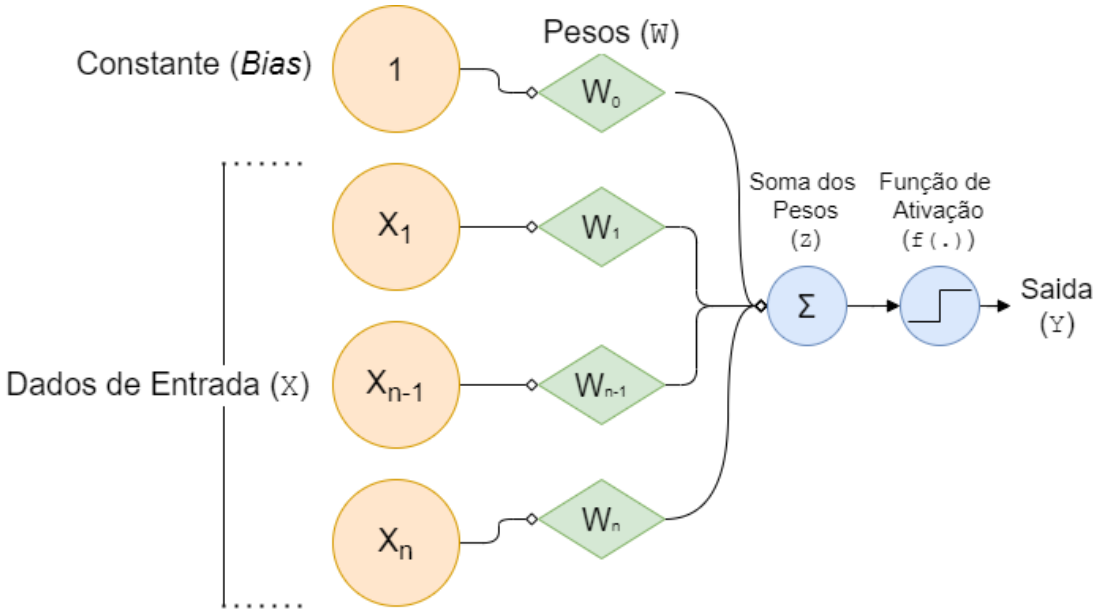


Figura 7. Arquitetura de uma rede neuronal simples de uma única camada (*Perceptron*).

O funcionamento de uma rede do tipo *perceptron* começa pela multiplicação de todos os valores de entrada pelos seus respetivos pesos. De seguida, todos os resultados obtidos pela multiplicação são somados para criar a primeira soma ponderada. O resultado da soma ponderada é transformado pela função de ativação, produzindo a saída do modelo *perceptron.* A função de ativação desempenha o papel integral de assegurar que a saída é mapeada entre os valores necessários, normalmente entre [0, 1] ou, em alguns casos entre [-1, 1]. O funcionamento de um *perceptron* pode ser expresso da seguinte forma: $Y = f(\sum_{i=0}^{n} X_i . W_i + W_0)$.

Ao longo da aprendizagem é importante perceber que o peso de determinada entrada é indicativo da importância da respetiva ligação. Da mesma forma, o valor do *bias* de uma entrada permite o deslocamento da curva da função de ativação, para cima ou para baixo. Quanto à função de ativação f(.), numa tarefa de classificação binária, esta é normalmente uma função *sigmoid* logística e é normalmente utilizada e é dada por: $\sigma(z) = 1/(1 + \exp(-z))$. No caso de classificadores de múltiplas classes, a função de ativação é normalmente a *Softmax*, onde os valores de saída podem ser interpretados como probabilidades. No caso da *Softmax*, a expressão matemática é dada por: $s(Z_k) = \exp(Z_k) / \sum_{i=0}^{k} \exp(z_i)$. Como forma simplificada de uma rede neuronal, especificamente uma rede neuronal de camada única, os *perceptrons* desempenham um papel importante em aplicações com recurso à classificação binária. No capítulo 2.4.5 pode ser encontrada uma revisão mais abrangente ao tópico das funções de ativação.

2.4.4.2 Rede Neuronal de Múltipla Camada

Uma das principais limitações da rede neuronal de camada única deve-se à sua característica linear em aplicações com classificadores, apesar desta utilizar uma função de ativação não linear. No entanto, esta limitação pode ser contornada através da introdução da chamada camada "oculta" entre a camada de entrada e a camada de saída, como verificado na Figura 8.

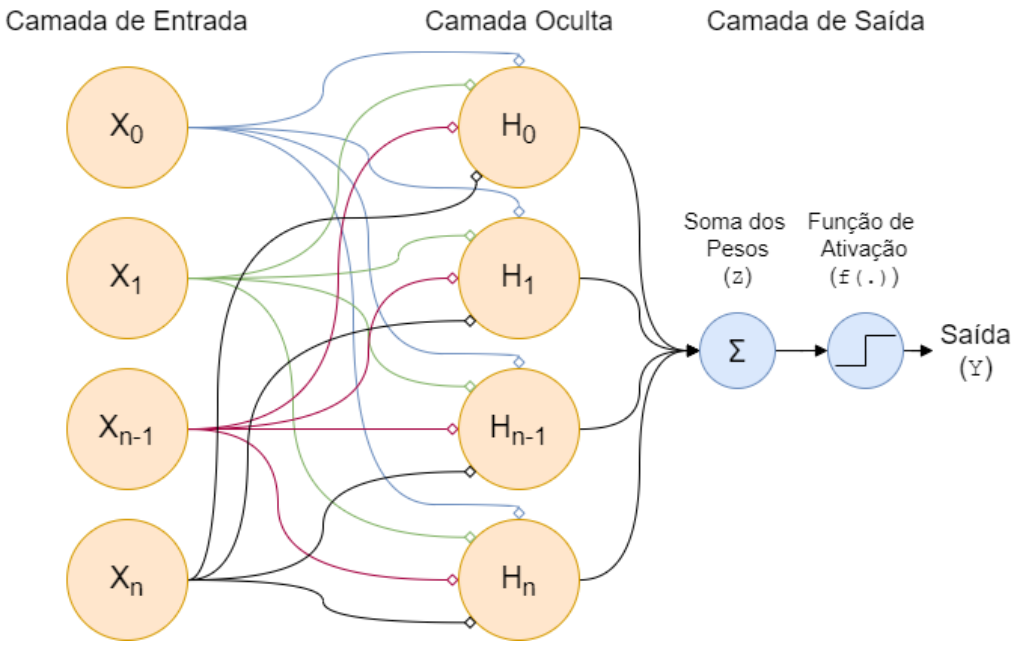


Figura 8. *Perceptron* com camada oculta.

Quanto à função de ativação, embora, em teoria, seja possível aplicar diferentes tipos de funções de ativação em diferentes camadas, ou até mesmo em diferentes unidades, é comum aplicar o mesmo tipo de função de ativação nas camadas ocultas. No entanto, deverá ser implementada uma função não linear caso contrário, a função será representada por uma rede neuronal de camada única com uma matriz de pesos, igual à matriz resultante da multiplicação da matriz de pesos das camadas ocultas. Normalmente, a função de ativação f(.) é definida com uma função do tipo *sigmoid* como, por exemplo, a *sigmoid* logística ou então uma função tangente hiperbólica ($\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$) devido às suas características não lineares e diferenciáveis. A diferença entre as funções de ativação descritas reside no intervalo de valores de saída, ou seja, os valores reais são ajustados para um intervalo compreendido entre [0, 1], no caso da utilização da função *sigmoid* logística, e para o intervalo entre [-1, 1], no caso da utilização da função tangente hiperbólica.

2.4.4.3 Processo de Treino de Redes Neuronais com *Backpropagation*

O processo de aprendizagem de uma rede neuronal tem por base um processo iterativo de constante otimização de pesos de forma a minimizar o erro. Com base no desempenho da rede, os pesos são modificados no conjunto de dados de exemplo/teste tendo como relação o conjunto de treino. Para além das várias etapas necessárias ao processo de aprendizagem, esta possui ainda mecanismos de recuo e de avanço. Durante o treino de uma rede neuronal, qualquer uma das funções de ativação no processo de propagação com avanço (*Feed-Forward Propagation*) é selecionada, sendo o treino com *backpropagation* utilizado para alterar os pesos. Redes neuronais FFNN com múltiplas camadas quando elaborados com implementação de *backpropagation* ganham uma ajuda extra na aprendizagem da relação entre as entradas e as saídas a partir das amostras introduzidas durante o processo de treino [99]. O processo de propagação para a frente e para trás, em uma *Deep Neural Network* (DNN), com apenas uma camada oculta é demonstrado no fluxograma da Figura 9.

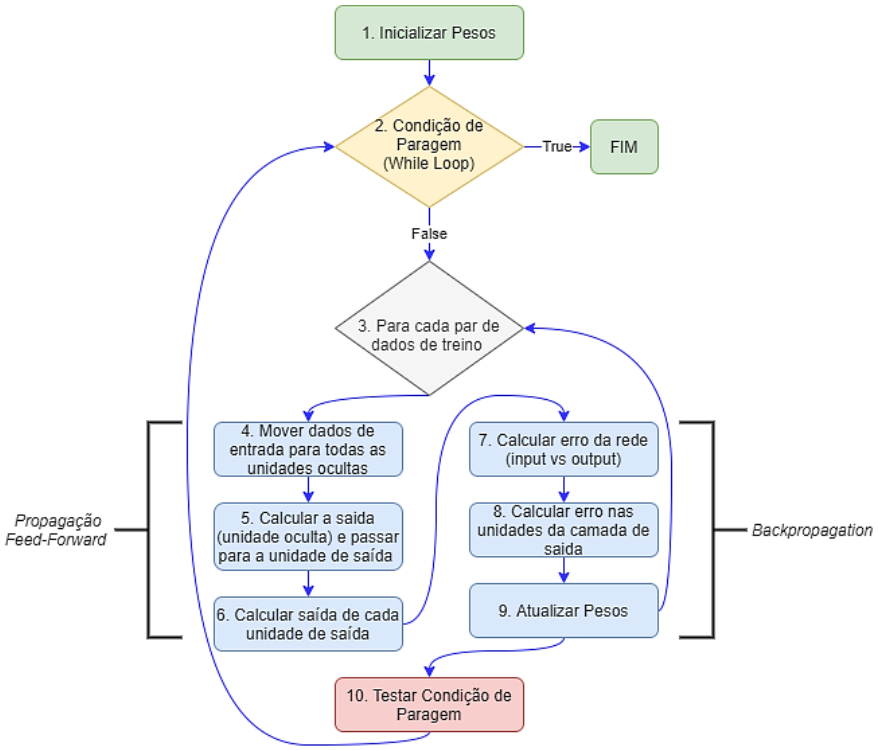


Figura 9. Processo de aprendizagem de uma DNN com *Feed-Forward Propagation* e *Backpropagation.*

1. Iniciar todos os pesos com valores aleatórios;
2. Ciclo até que a condição de paragem definida seja cumprida;
3. Ciclo iterativo para cada par do conjunto de dados de treino $((x_1, y_1) \dots (x_{n,} y_n))$ (ciclo contempla os passos 4, 5, 6, 7, 8 e 9);
4. Cada unidade de entrada $(X_i, i = 1, 2, 3, \dots, n)$ recebe o sinal de entrada X_i e envia este mesmo sinal a todas as unidades da camada oculta;
5. Cada unidade da camada oculta $(Z_j, j = 1, 2, 3, \dots, p)$ calcula a saída usando $Z_{j\_input} = b + b_j + \sum_{i=1}^{n} w_{ij} \,.\, x_i$, transmitindo o resultado à unidade de saída por aplicação da função de ativação $Z_j = f(Z_{j\_input})$;
6. Calcular o sinal de saída $(Y_k, k = 1, 2, 3, \dots, m)$ para cada unidade da última camada usando $Y_{k\_input} = b_k + \sum_{j=1}^{p} z_j \,.\, w_{jk}$ e por fim calcular a ativação com $Y_k = f(Y_{k_{input}})$;
7. Para cada par do conjunto de dados de treino, na entrada $(X_1, X_2, \dots, X_n)$ e correspondente par de saída $(Y_1, Y_2, \dots, Y_n)$ seja $t_1, t_2, \dots, t_n$ o alvo padrão. Em cada saída, o neurónio calcula o erro da rede $(\delta_k = (t_k - Y_k) f(Y_{k\_input})$;
8. Em cada neurónio da camada oculta, calcular o respetivo erro (δ_j) usando o erro dos neurónios de saída obtidos na etapa anterior (δ_k) $(\delta_j = f'\left(z_{j\_input}\right) \sum_k^m \delta_k \,.\, w_{jk})$;
9. Atualizar os pesos e *bias* usando as seguintes formulas, considerando η a taxa de aprendizagem.
    a. cada camada de saída $(Y_k, k = 1, 2, 3, \dots, m)$ atualiza os seus pesos $(J = 0, 1, 2, \dots, P)$ $(W_{jk}(new) = W_{jk}(old) + \eta \,.\, \delta_k . Z_j)$ e *bias* $(b_k(new) = b_k(old) + \eta \,.\, \delta_k)$;

b. cada camada oculta $(Z_J, J = 1, 2, 3, \dots, p)$ atualiza os seus pesos $(i = 0, 1, 2, \dots, n)$ $(W_{ij}(new) = W_{ij}(old) + \eta \,.\, \delta_j . X_i)$ e *bias* $(b_j(new) = b_j(old) + \eta \,.\, \delta_j)$;

10. Verificar a condição de paragem.

A *backpropagation* é um dos métodos mais utilizado durante a aprendizagem de uma rede neuronal artificial [100]. A sua utilização é comum juntamente com um método de otimização denominado por gradiente de descida. Este é um algoritmo de otimização que é usado para encontrar o mínimo local de uma função, o custo ou erro. São três as principais variações deste gradiente, que diferem na quantidade de dados que utilizamos para calcular o gradiente da função objetivo. Dependendo desta quantidade de dados, é efetuada uma troca entre a precisão da taxa de atualização do parâmetro e o tempo que este leva a realizar uma atualização [101]. As três variações são brevemente descritas de seguida:

Batch Gradient Descent

Também conhecido por gradiente de descida de "baunilha", esta variação é a mais usada normalmente. Este calcula o gradiente da função de perda em relação aos parâmetros θ da seguinte forma: $\theta = \theta - \eta \,.\, \nabla_\theta . L(\theta)$ [101]. Dada a necessidade de calcular os gradientes de todo o conjunto de dados para realizar apenas uma atualização, o *batch* gradiente de descida torna-se muito lento, e intratável para conjunto de dados que excedam limites de memória.

Stochastic Gradient Descent (SGD)

Esta variante em vez de passar todo o conjunto de dados de treino por uma única atualização, passa uma atualização de parâmetros por cada amostra de treino $X^{(i)}$ com a respetiva classificação $Y^{(i)}$ [101]. A função matemática da variante SGD é expressa da seguinte forma: $\theta = \theta - \eta \,.\, \nabla_\theta J\,(\theta; X^{(i)}\,; Y^{(i)})$.

Esta abordagem resolve o problema encontrado na variante gradiente de descida em lote, quanto ao número necessário de cálculos por atualização. O SGD elimina esta redundância concluindo uma atualização de cada vez, permitindo assim ser geralmente mais rápido e evitando o cálculo de valores semelhantes em cada atualização de parâmetros. No entanto, deverá ser dada consideração ao facto da utilização desta variante poder levar a flutuações na minimização do objetivo da função.

Mini-batch Gradient Descent

Esta variante, trata-se de um meio termo entre as duas anteriores variantes. Esta variante em vez de atualizar para cada amostra de treino, realiza as atualizações em pequenas (*mini-batch*) instâncias do conjunto de dados de treino. A função matemática pela qual é possível expressar esta variante é conseguida por: $\theta = \theta - \eta \,.\, \nabla_\theta L\,(\theta; X^{(i:i+n)}\,; Y^{(i:i+n)})$. Ao escolher um pequeno lote, ou mini-batch, esta abordagem reduz as flutuações das atualizações, que consistia na desvantagem da variante SGD. Considerando uma época (*epoch*) a passagem de todas as instâncias do conjunto de dados de treino pelo algoritmo e uma iteração ser dada por concluída após cada mini lote ser

processado, então o número de iterações em cada época dependerá do tamanho do mini lote (*mini-batch*).

2.4.5 Funções de Ativação

A função de ativação consiste em um mecanismo pelo qual os neurónios artificiais processam e transferem informação [102]. As funções de ativação são utilizadas em redes neuronais, para calcular a soma ponderada dos dados de entrada e do *bias*, permitindo posteriormente decidir se um neurónio pode ou não ser ativado (mecanismo de ativação é também conhecido por disparo do neurónio) [103]. Existem diferentes tipos de funções de ativação que podem ser utilizadas com base nas características da aplicação. Estas, podem ou não ser lineares, dependendo da função que representam, e são continuamente diferenciáveis. A propriedade de diferenciação é pertinente, principalmente quando usado juntamente com o gradiente de descida.

No caso de um modelo linear, o mapeamento linear de uma função de entrada para a sua saída, tal como realizado nas camadas ocultas antes da previsão final de uma classe, é conseguido na maioria dos casos, com recurso à transformação de *affine* [102]. As redes neuronais movem dados através de uma rede interna que, em cada camada, realiza uma transformação linear (também referida como transformação *affine*) dos seus dados de entrada, seguida de uma transformação não linear através da função de ativação [94, 104, 105].

Algumas das funções de ativação amplamente utilizadas em arquiteturas estão listadas na Tabela 1.

Tabela 1: Funções de ativação.

No.	Nome da função [Referência]	Função Matemática
1	*Sigmoid* [106, 107]	$f(x) = \frac{1}{1 + exp^{-x}}$
2	*Hyperbolic Tangent* (*Tanh*) [108, 109]	$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$
3	*Softmax* [94]	$f(x) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$
4	*Softsign* [107]	$f(x) = \frac{x}{\lvert x \rvert + 1}$
5	*Rectified Linear Unit* (*ReLU*) [110]	$f(x) = \max(0, x) = \begin{cases} x_i, & x_i \geq 0 \\ 0, x_i < 0 \end{cases}$
6	*Softplus* [111]	$f(x) = \log(1 + exp^{x})$
7	*Exponencial Linear Units* (*ELUs*) [112]	$f(x) = \begin{cases} x, & x > 0 \\ \alpha \exp(x) - 1, x \leq 0 \end{cases}$
8	*Maxout* [113]	$f(x) = \max(w_1^T x + b_1, w_2^T x + b_2)$

Tabela 1 (cont.). Funções de ativação.

9	*ELiSH* [114]	$f(x) = \begin{cases} \frac{x}{1 + e^{-x}}, & x \geq 0 \\ \frac{e^{x} - 1}{1 + e^{-x}}, & x < 0 \end{cases}$
10	*Swish* [115]	$f(x) = x \,.sigmoid(x) = \frac{x}{1 + e^{-x}}$

Função *Sigmoid* (Tabela 1, No. 1), também conhecida como função logística, possui três variantes amplamente usadas em DL, *Hard Sigmoid, Sigmoid-Weighted Linear Units* e *Derivative of Sigmoid-Weighted Linear Units*. A função *Sigmoid* é uma função de ativação não linear utilizada maioritariamente em redes neuronais do tipo *feed-forward* [107]. A *Sigmoid* é uma função diferenciável limitada, definida para valores de entrada reais, valores positivos resultado das derivadas e algum grau de suavidade [116]. A função *Sigmoid* é utilizada nas camadas de saída, sendo utilizada para prever a saída, com base na probabilidade em problemas de classificação binária e tarefas de regressão logística. Uma das desvantagens da *Sigmoid* está associada à problemática conhecida por *vanishing gradient*, que está relacionada com o processo de treino de uma rede neuronal, onde são utilizados métodos de aprendizagem baseados em gradientes e *backpropagation*. À medida que se adicionam mais camadas em determinado modelo, com uso de determinadas funções de ativação, os gradientes da função de perda aproximam-se de zero, dificultando o processo de treino.

A função *Tanh* (Tabela 1, No. 2) é também amplamente utilizado em DL e possui variantes como, por exemplo, a função *Hard Hyperbolic*. A *Hyperbolic Tangent* é uma função de suavização com o objetivo de centragem em zero [117], cujo intervalo de valores se situa entre [-1, 1] [109]. Apesar da *Tanh* ser mais utilizada que a *Sigmoid*, devido ao melhor desempenho no treino em arquiteturas de múltiplas camadas [118], esta, tal como a *Sigmoid,* possuem como desvantagem o problema de *vanishing gradient*. A principal vantagem consiste na característica da centragem em zero, o que por si só colabora com a utilização dos processos de *backpropagation.* Na Figura 10 é possível verificar as respostas obtidas com o uso das funções de ativação *Sigmoid* e da *Tanh.*

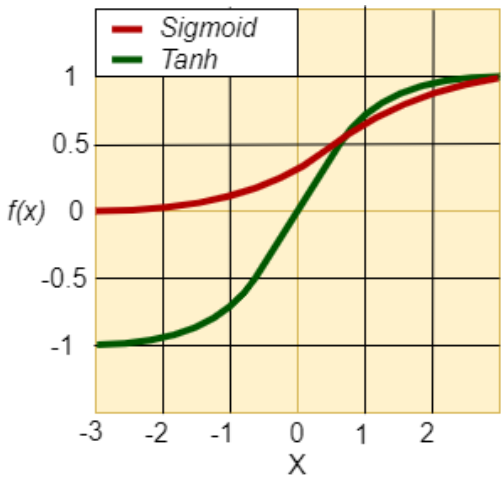


Figura 10. Respostas característica das funções de ativação: *Sigmoid* e *Hyperbolic Tangent.*

A função *Softmax* (Tabela 1, No. 3) é utilizada para calcular a distribuição de probabilidade a partir de um vetor de números reais. Esta produz uma saída cujo intervalo de valores está compreendido entre [0, 1], sendo a soma das probabilidades igual a 1. A *Softmax* é usada em modelos de múltiplas classes, onde esta devolve as probabilidades de cada uma das classes, tendo como classe alvo a que tiver a maior probabilidade. A *Softmax*, quando usada em arquiteturas de modelos de DL, é usada em quase todas as camadas de saída [119, 120]. A principal diferença entre as funções de ativação *Sigmoid* e *Softmax* é o fato da *Sigmoid* ser usada em tarefas de classificação enquanto a *Softmax* é usada em tarefas de classificação multivariada.

A *Softsign* (Tabela 1, No. 4) é um polinómio quadrático, função não linear introduzida por Turian *et al.* (2009) [107], também amplamente utilizada em arquiteturas DL. Apesar de usada maioritariamente em tarefas de regressão [121] tem também sido usada em estudos de DL nas áreas de voz e fala apresentando resultados promissores [122].

A função *ReLU* (Tabela 1, No. 5), função de ativação de unidade linear retificada, foi proposta por Nair *et al.* (2010) [110]. A *ReLU* oferece melhor desempenho e generalização no processo de treino em modelos de DL, quando comparado com funções de ativação como a *Sigmoid* e a *Tanh* [123, 124], sendo por isso atualmente uma preferência na escolha de funções de ativação [115]. A *ReLU* apresenta-se como uma função quase linear, preservando as propriedades dos modelos lineares que se tornaram fáceis de otimizar devido aos métodos de gradiente de descida [94]. Esta função retifica os valores inferiores a zero, forçando-os a ser zero, eliminando o problema *vanishing gradient*, descrito em algumas das funções de ativação anteriores. A vantagem de maior destaque do uso da *ReLU* diz respeito ao processamento rápido, devido ao facto de não serem calculados exponenciais nem divisões [123]. A *ReLU* possui ainda as seguintes variantes: *Leaky ReLU* (LReLU), *Parametric Rectified Linear Units* (PReLU) e *Randomized Leaky ReLU* (RReLU). Na Figura 11 é possível comparar a resposta característica das funções de ativação *ReLU* com a *Sigmoid* e a *Tanh.*

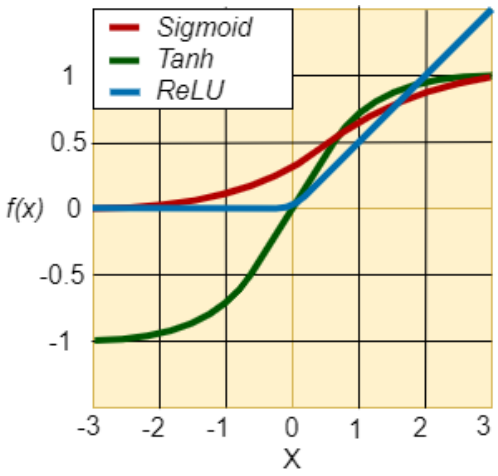


Figura 11. Respostas característica das funções de ativação: *Sigmoid,Tanh e ReLU.*

A função de ativação *Softplus* (Tabela 1, No. 6) possui, tal como a *ReLU*, propriedades de suavização e gradiente não nulo, com bons resultados na estabilização e desempenho geral da rede neuronal. A *Softplus* foi proposta por Dugas *et al.* (2001) [125] e pretende ser uma versão mais suave da *ReLU*, com a capacidade de limitar a saída a um valor sempre positivo.

As unidades lineares exponenciais (Tabela 1, No. 7), *ELU's,* foram propostas por Clevert *et al.* (2015) [112] e são utilizadas para acelerar o processo de treino de modelos de DL. A principal vantagem das *ELUs* deve-se à possibilidade de estas aliviarem o problema *vanishing gradient* utilizando a identidade para valores positivos, e também melhorar as características da aprendizagem. Permitem valores negativos, o que possibilita mover a ativação da unidade média para mais perto de zero, reduzindo assim a complexidade computacional e aumentando a velocidade do processo de treino [112]. A *Parametric Exponencial Linear Unit* (PELU) [126] e a *Scaled Exponencial Linear Units* (SELU) [127] são exemplos de variantes das *ELUs.*

A função de ativação *Maxout* (Tabela 1, No. 8) é uma função onde a característica não linear é aplicada como um produto entre os pesos de uma rede neuronal e os seus dados. Proposta por Goodfellow *et al.* (2013), a *Maxout* representa uma generalização das funções *ReLU* e *LReLU* [113]. Esta consiste em uma função linear dividida em partes que devolve o valor máximo das entradas. A principal desvantagem da *Maxout* deve-se ao grande poder computacional necessário, dado que esta duplica o número de parâmetros em cada neurónio.

A função de ativação *ELiSH* (Tabela 1, No. 9), *Exponencial Linear Squashing*, foi proposta recentemente por Basirat *et al.* (2018) [114]. É composta pelas funções *ELU* e Sigmoid, partilhando ainda propriedades em comum com a função *Swish*. As propriedades da *ELiSH* variam tanto nas partes negativas como nas positivas, conforme definido pelos seus limites. A componente partilhada *Sigmoid* melhora o fluxo de informação, enquanto as partes lineares eliminam a problemática descrita como *vanishing gradient*. O conjunto de dados *ImageNet* [128] juntamente com a utilização de arquiteturas de DL, onde a função *ELiSH* é implementada, têm sido utilizados com sucesso [114]. A variante mais conhecida da função *ELiSH é a HardELiSH que* contempla a multiplicação da

HardSigmoid com a *ELU* na parte negativa, e a multiplicação da *Linear* com a *HardSigmoid* na parte positiva [114].

A função *Swish* (Tabela 1, No. 10) é uma das primeiras funções de ativação híbridas, composta pela combinação da função *Sigmoid* com a própria função de entrada. Esta foi proposta por Ramachandran *et al.* (2017) [115] e utiliza a técnica de procura automática baseada na aprendizagem de reforço (*reinforcement learning*) para calcular a função. As propriedades da função *Swish* incluem suavização, limitação apenas na parte inferior da função e não monotónica. A propriedade de suavização permite à *Swish* produzir melhores resultados de otimização e generalização, quando utilizada no processo de treino de arquiteturas de DL [115]. As principais vantagens da função *Swish* dizem respeito à sua simplicidade e maior precisão, dado que esta não sofre de *vanishing gradient*, fornecendo uma boa propagação de informação durante o processo de treino. Segundo Ramachandran *et al.* (2017) a utilização da função *Swish* apresenta melhores resultados quando comparados com a utilização da *ReLU* em tarefas de classificação em modelos de DL [115].

Alguns exemplos de funções de ativação, com referência à posição e arquitetura onde estas são utilizadas, podem ser encontrados na Tabela 2.

Tabela 2: Exemplos de Arquiteturas com referência às funções de ativação utilizadas.

Arquitetura	Camadas Ocultas	Camada de Saída	Referência
SeNet	ReLU	Sigmoid	[129]
MobileNets	ReLU	Softmax	[130]
ResNeXt	ReLU	Softmax	[131]
ResNet	ReLU	Softmax	[132]
SqueezeNet	ReLU	Softmax	[133]
GoogleNet	ReLU	Softmax	[134]
SegNet	ReLU	Softmax	[120]
VGGNet	ReLU	Softmax	[135]
ZFNet	ReLU	Softmax	[136]
NiN	Sem ativação	Softmax	[137]
AlexNet	ReLU	Softmax	[119]

2.4.6 Algoritmos de *Deep Learning*

Deep Learning (DL) é um subdomínio da área de *Machine Learning* com o intuito de desenvolver arquiteturas artificiais inspiradas nas redes neuronais biológicas presentes no cérebro humano. Uma rede neuronal artificial pretende, imitando o cérebro humano, processar e aprender com a informação disponibilizada. À semelhança do cérebro humano, os neurónios artificiais estão ligados entre si, tal como as sinapses ligam a saída de um dos neurónios à entrada de um outro (Figura 12). Estas linhas que unem os neurónios possuem ligações com informação numérica denominada por pesos que representam a força que as ligações descritas possuem entre si. Os pesos referidos podem ser parametrizados com base na experiência, tornando as redes neuronais capazes de aprender com base em entradas fornecidas, ou seja, com base em uma instância de um determinado conjunto de dados.

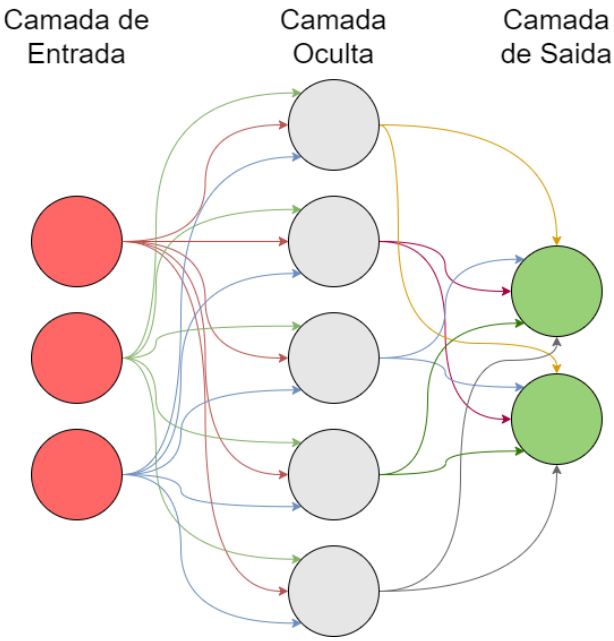


Figura 12. Rede Neuronal *Feed-forward* simples.

A melhor definição de uma rede neuronal profunda, dado que não existe um consenso quanto à sua definição, é de que esta pode ser vista como qualquer rede neuronal artificial possuindo duas ou mais camadas ocultas. Ao contemplarem várias camadas, estas redes são capazes de aprender representações dos dados com múltiplos níveis de abstração [94]. Nos últimos anos esta abordagem tem ganho muito interesse devido ao facto de superar as metodologias comuns utilizadas até então para classificação com uso de vários conjuntos de dados [138] e evoluído em vários campos, desde o reconhecimento de voz e imagem [109] e também na descoberta de drogas e nos domínios da genética [139].

2.4.6.1 Autoencoder

Os auto codificadores (AE) são arquiteturas de DL do tipo *feed-forward* e o seu funcionamento exemplifica o princípio da aprendizagem não supervisionada de representação [140]. O AE é útil quando usado em conjuntos de dados em que os dados sem etiqueta, sem rótulo, são superiores aos dados com etiqueta. A arquitetura da rede divide-se na zona de codificação, onde a AE codifica determinada entrada X num espaço de dimensão inferior Z, e na zona de descodificação, onde a representação codificada é novamente descodificada numa representação aproximada X' da entrada X através da camada oculta Z da zona de *encoding* (Figura 13) [141].

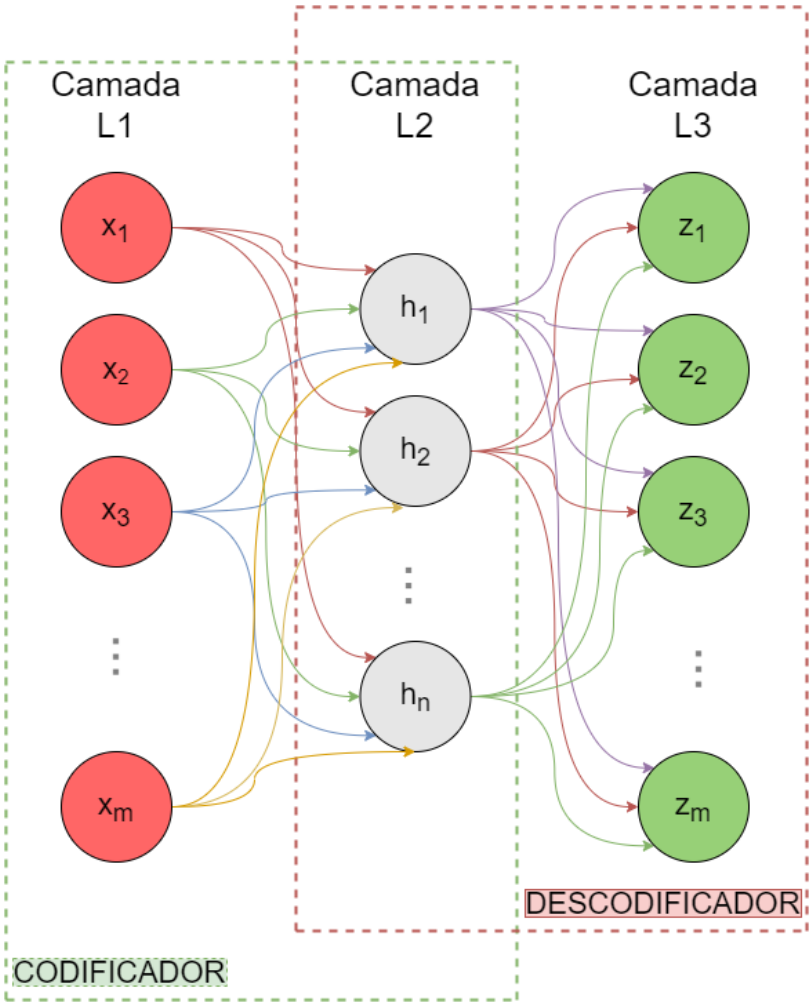


Figura 13. Autoencoder (AE).

O princípio básico de funcionamento do AE consiste em três passos: codificação, descodificação e cálculo do erro quadrático.

1. No processo de codificação, é convertido o vetor de entrada $x \in \Re^m$ em $h \in \Re^n$, a camada oculta por $h = f(wx + b)$ onde $w \in \Re^{m.n}$ e $b \in \Re^{n.m}$ onde n são as dimensões do vetor de entrada convertidos em estado oculto. A dimensão da camada oculta h deve ser menor que a dimensão em x, e f representa a função de ativação.
2. O processo de descodificação reconstrói, descodifica, com base em h, o vetor de entrada z com base na equação $z = f'(w'h + b')$ onde $w' \in \Re^{n.m}$ e $b' \in \Re^m$. De forma semelhante ao passo anterior, f' representa a função de ativação.
3. Por fim, é calculado o erro quadrático por $L_{recons}(x, z) = ||x - z||^2$, que consiste na função de custo do erro de reconstrução ou descodificação. A minimização do erro de reconstrução é conseguida através da otimização da função de custo: $J(\Theta) = \sum(x, z), \ \Theta = \{w, w', b, b'\}$.

Os *Autoencoder's* são então considerados modelos não supervisionados já que estes não necessitam que os dados introduzidos estejam explicitamente rotulados (dados com etiqueta presente). No entanto, os AE podem ser considerados auto-supervisionados já que estes geram os

seus próprios rótulos a partir dos dados de formação, entre o processo de codificação e descodificação.

Uma variante do AE conhecida por autocodificador empilhado (SAE) consiste em mais um modelo de aprendizagem não supervisionada. A arquitetura de um SAE é composta por um conjunto de camadas de AE's dispostas umas em cima das outras, onde a camada de saída de cada camada AE é ligada às entradas da camada seguinte (Figura 14).

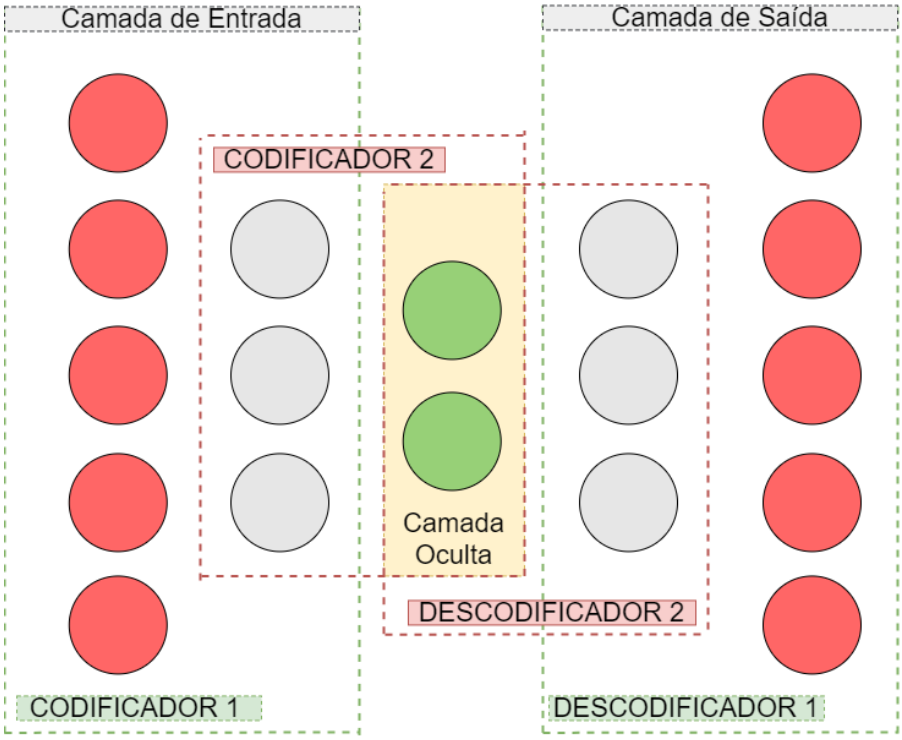


Figura 14. Autoencoder SAE.

O AE do tipo *Denoising* (DAE) foi introduzido por Vincent *et al.* (2010) [142] e o seu processo de treino pretende reconstruir a informação de entrada a partir desses mesmo dados mas com a adição de ruído aleatório. No caso do AE variável (VAE), o objetivo é modificar o *encoder* onde o espaço vetorial latente é utilizado para representar as imagens que seguem um unidade de distribuição *gaussiana* [143]. Dependendo da variante de *Autoencoder*, estas arquiteturas podem ser utilizadas em aplicações de aprendizagem supervisionada, não supervisionada e em aplicações de segmentação [144].

2.4.6.2 Máquina de *Boltzmann* Restrita

A arquitetura de uma máquina restrita de *Boltzmann* (RBM) é composta por um *Markov Random Field* (MRF) associado ao modelo generativo probabilístico de duas camadas não direcionadas como ilustrado na Figura 15.

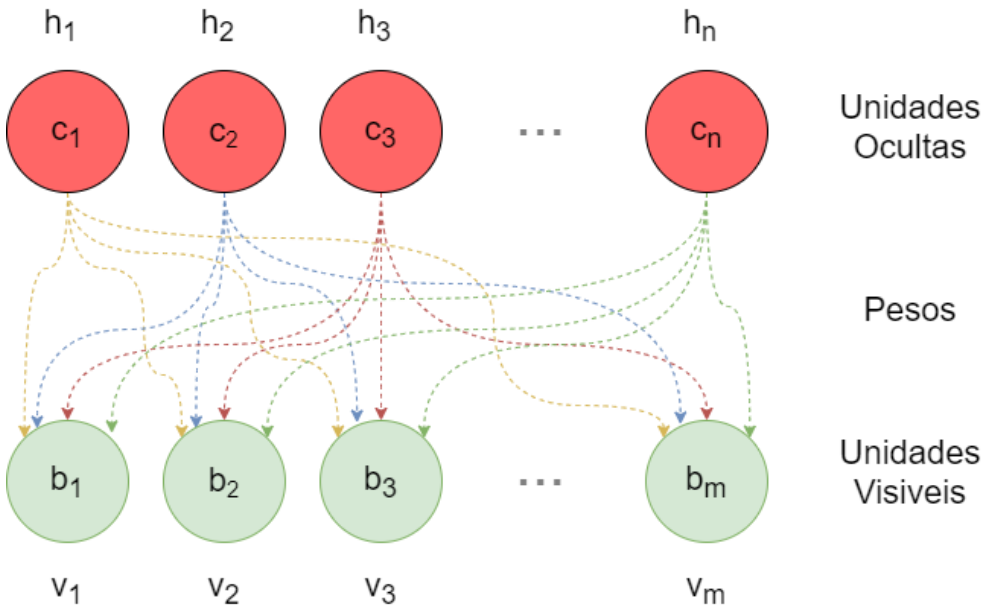


Figura 15. Máquina de *Boltzmann* restrita com n unidades ocultas e m unidades visíveis.

A RBM contém unidades visíveis na entrada (v) e unidades ocultas na saída (h). A arquitetura RBM é caracterizada pela falta de contacto direto entre as duas unidades visíveis ou entre as duas camadas ocultas. No caso de RBM binárias, as variáveis aleatórias (v, h) ficam $(v, h) \in \{0,1\}^{m+n}$. Tal como a genérica máquina de *Boltzmann*, as RBM são modelos baseados em energia [145], onde o estado desta $\{v, h\}$ é definido por $E\,(v, h) = -\sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij} h_i v_j - \sum_{j=1}^{m} b_j v_j - \sum_{i=1}^{n} c_i h_i$. Os elementos (v_j, h_i) são os estados binários da unidade visível $j \in \{1, 2, \dots, m\}$ e da unidade oculta $i \in \{1, 2, \dots, n\}$, enquanto b_j, c_i são as *bias's* das unidades visíveis e ocultas. O termo de interação simétrica entre unidades v_j e h_i é representado por w_{ij} [146]. Os parâmetros (w_{ij}, b_j, c_i) são calculados eficientemente utilizando o método de aprendizagem por divergência contrastiva [147] ou uma versão em lote da divergência contrastiva em k-passos (CD-k) [148].

2.4.6.3 Deep Belief Networks

A arquitetura *Deep Belief Network* (DBN), proposta por Hinton *et al.* (2006), representa modelos generativos probabilísticos compostos por múltiplas camadas de variáveis estocásticas e latentes [149]. Estas são um modelo não convolutivo com capacidade de extrair características e consequentemente aprender uma representação hierárquica profunda dos dados oriundos da formação. A arquitetura das DBN é considerada híbrida, sendo composta por múltiplas RBM formando assim um modelo generativo dirigido (Figura 16). As variáveis latentes possuem tipicamente valores binários e são muitas vezes chamadas de unidades ocultas servindo como detetores de características.

Os modelos generativos fornecem uma distribuição conjunta de probabilidade sobre o conjunto de dados de entrada e respetivas etiquetas, facilitando a estimativa $P(x|y)$ tal como a $P(y|x)$, enquanto os modelos discriminatórios utilizam apenas o ultimo modelo $P(y|x)$ [150].

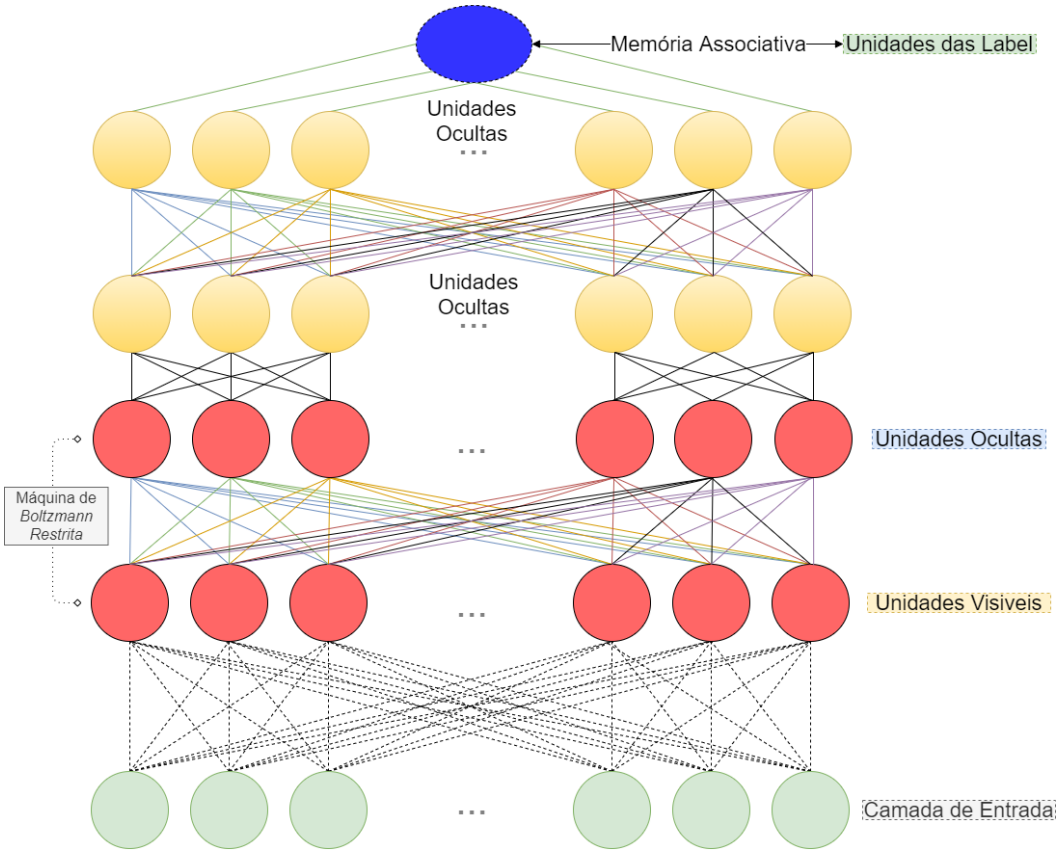


Figura 16. *Deep Belief Networks.* Adaptado de Al-Jabery *et al.* (2019) [150].

Como ilustrado na Figura 16 as DBN são compostas por várias camadas de redes neuronais, especificamente as máquinas de *Boltzmann* restritas. Cada uma destas está restrita a uma única camada visível e a uma camada oculta. As camadas superiores possuem ligações simétricas e não direcionáveis entre si formando uma memoria associativa com as unidades das etiquetas. As camadas inferiores recebem ligações de cima para baixo, dirigidas a partir da camada superior, enquanto os estados das unidades na camada inferior representam a camada de entrada do vetor de dados.

O modelo DBN de distribuição conjunta entre as unidades observadas v e as camadas ocultas l $(h^k(k = 1, \dots, l))$ é conseguido por $P(v, h^1, \dots, h^l) = \left(\prod_{k=0}^{l-2} P\left(h^k \middle| h^{(k+1)}\right)\right) P\left(h^{(l-1)}, h^1\right)$, onde $v = h^0, P(h^k|h^{(k+1)})$ é uma distribuição condicional dada por $\left[P\left(h_i^k = 1 \middle| h^{(k+1)}\right) = \sigma\left(b_i^k + w_{:,1}^{(k+1)} h^{(k+1)}\right) \forall i, \forall k \in 0, 1, \dots, \mathrm{l} - 2\right]$, para a camada k dadas as unidades de $k + 1$. As DBN possuem ainda l matrizes de pesos: $w^1, \dots, w^l$ e $l + 1$ vetores de *bias*: $b^0, \dots, b^l$. A distribuição de probabilidade da DBN é dada por $P(v_i = 1|h^1) = \sigma(b_i^0 + w_{:,1}^{(1)} h^1) \forall i$.

As DBN foram desenvolvidas como solução para alguns dos problemas ou dificuldades encontradas em redes neuronais tradicionais, tais como a aprendizagem lenta, insuficiente seleção de parâmetros que origina bloqueios nos mínimos locais e a necessidade de muitos dados de formação para o processo de aprendizagem.

2.4.6.4 Redes Neuronais Convolucionais

Redes neuronais convolucionais (CNN) representam um tipo específico de redes neuronais especializado em processamento de dados, onde estes possuem uma topologia conhecida [151], como, por exemplo, tabelas de dados temporais (*time-series*) com uma topologia 1D e imagens com dados em topologia 2D [152]. As CNN representam uma arquitetura de rede neuronal, que utiliza a operação matemática convolução em vez da multiplicação, em pelo menos uma das suas camadas. A convolução é uma operação matemática especializada em operações lineares. Dentro das diferentes famílias de arquiteturas de redes neuronais em DL, as CNN são uma família única de arquiteturas especificas amplamente usadas para a identificação e caracterização de padrões em imagens. O desenvolvimento destas emerge principalmente dos estudos do *córtex* visual encontrado nos seres vivos [153, 154]. Um dos problemas encontrados em algumas redes neuronais (por exemplo modelos do tipo *fully connected feed-forward*) é o número crescente de parâmetros mesmo em arquiteturas pouco "profundas" onde existem inúmeros neurónios fazendo com que estes modelos se tornem impraticáveis em aplicações com imagem. Ao utilizar um modelo baseado em CNN é possível elaborar arquiteturas mais "profundas" mesmo com um número reduzido de parâmetros.

A arquitetura de uma CNN está alicerçado em três conceitos, pesos partilhados, campos recetivos locais e a subamostragem espacial [151]. A operação de convolução permite a manipulação de dados não estruturados fazendo desta um elemento essencial na arquitetura e uma característica de distinção. A arquitetura tradicional de uma CNN (Figura 17) é composta pelas camadas convolucionais (CONV), camadas de *pooling* (POOL) e pela camada de ligação completa (FC), sendo a camadas CONV e POOL possíveis de ajustar com um conjunto de parâmetros (*hyperparameters*).

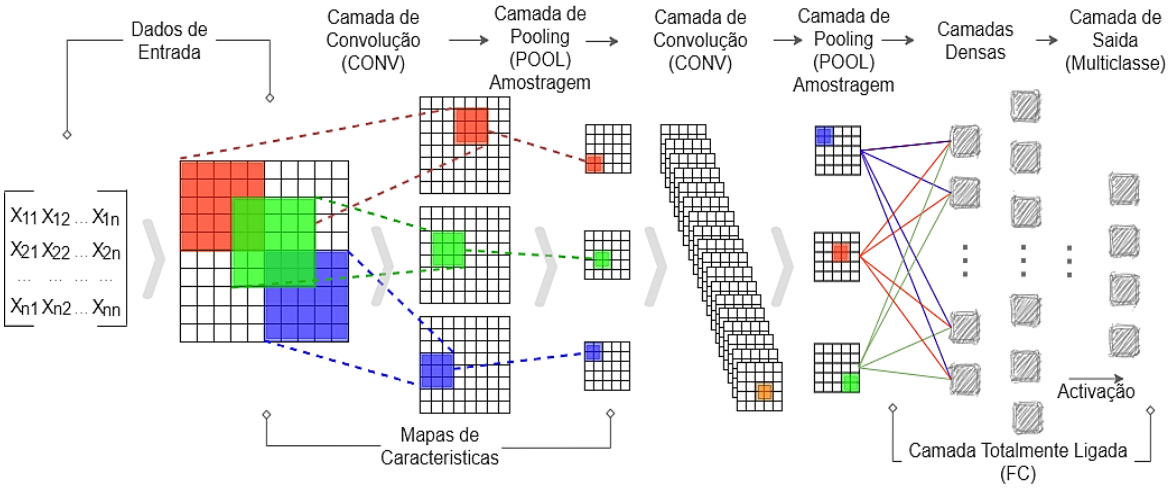


Figura 17. Arquitetura tradicional de uma CNN. Adaptado de Maeda-Gutierrez *et al.* (2020) e Albelwi *et al.* (2017) [155, 156].

A camada de convolução utiliza filtros que realizam operações de convolução ao analisar a entrada (*input*), tendo em consideração as suas dimensões. Os seus parâmetros de ajuste incluem o

tamanho do filtro (F) e o passo (*stride: S*). A saída resultante (O) é denominada por mapa de características ou mapa de ativação.

Camada de Convolução

A convolução do sinal de entrada $x(t)$ com o filtro $h(t)$ cria um sinal de saida $y(t)$ na tentativa de este revelar mais informação que o próprio sinal de entrada. A convolução de um sinal discreto $x(t)$ e $h(t)$ em dados com formato 1D é conseguida por $y(t) = x(t).h(t) = \sum_{t=\infty}^{\infty} x(\tau)h(t-\tau)$. No caso de imagens digitais $x(n_1, n_2)$ em que o sinal discreto é do tipo 2D, a convolução de $x(n_1, n_2)$ e $h(n_1, n_2)$ é dada por $y(n_1, n_2) = \sum_{k_1=0}^{M-1} \sum_{k_2=0}^{N-1} x(k_1, k_2)h(n_1 - k_1, n_2 - k_2)$, onde $0 \leq n_1 \leq M-1$ e $0 \leq n_2 \leq N-1$. A função da camada de convolução (Figura 18) é a de detetar características locais x^l a partir de mapas de características oriundos da entrada x^{l-1} usando *kernel's* k^l através das operações matemáticas de convolução.

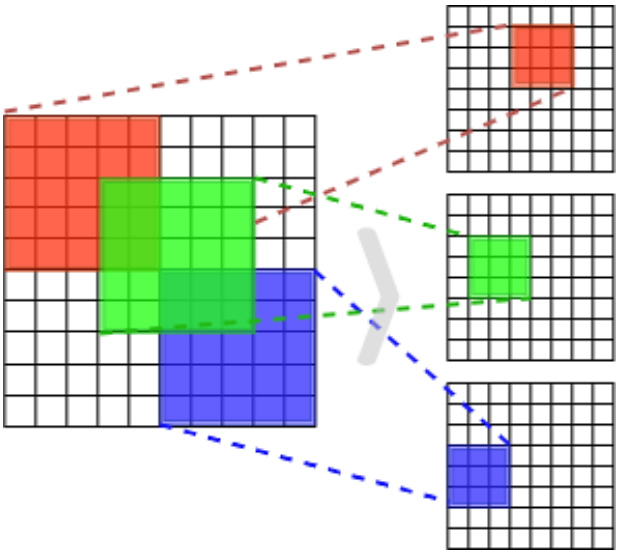


Figura 18. Operação de Convolução.

Esta operação de convolução é repetida em cada camada convolutiva sujeita a transformações não lineares através de $x_n^l = f(\sum_m^{M^{(l-1)}} x_m^{(l-1)} . k_{mn}^l + b_m^l)$ onde:

- k_{mn}^l representa os pesos entre o mapa de características m na camada $l-1$,
- o mapa de características n na camada $l.x_m^{(l-1)}$ representa o mapa de características m da camada $l-1$,
- x_n^l é o elemento n do mapa de características da camada $l.b_m^l$ sendo este o parâmetro de *bias,*
- a função de ativação não linear é representada por $f(.)$
- e por fim, $M^{(l-1)}$ denota o conjunto de mapas de características.

As CNN reduzem significativamente o número de parâmetros, quando comparado com modelos como as FCNN (*Fully Connected Neural Networks*), devido à ligação local e à partilha de pesos. A

profundidade, o preenchimento zero e os *strides* são três dos mais importantes hiper-parâmetros usados para controlar o volume da saída da camada de convolução.

Camadas de *Pooling*

A camada de *pooling* está disposta após a camada convolutiva de forma a subamostrar o mapa de características, com o objetivo de criar um estado de invariância espacial, de forma a minimizar a dimensão espacial dos mapas de características para a próxima camada de convolução (Figura 19).

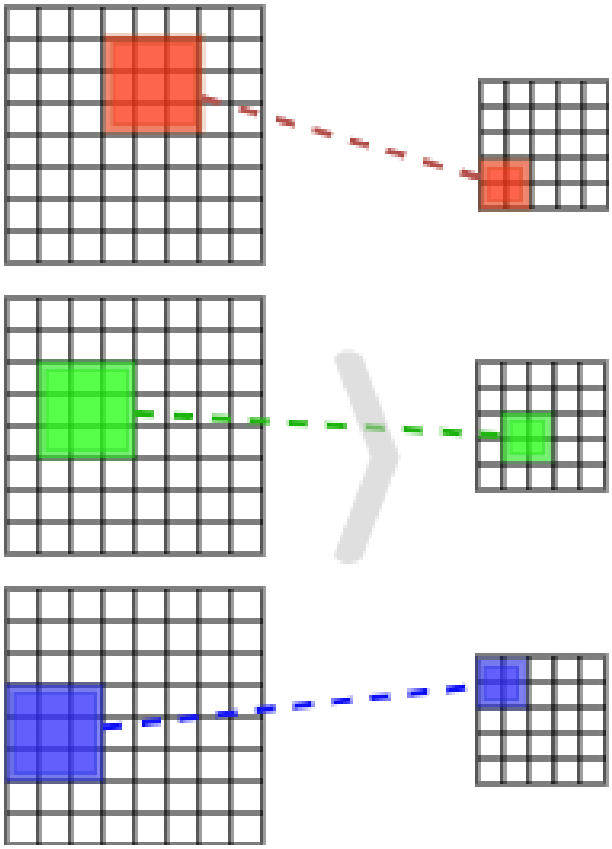


Figura 19. Operação de *Pooling.*

O *pooling* máximo ($x_i = max_{1 \le j \le M.M}(x_j)$) e médio ($x_i = {}^{1}/_{M.M} \sum_{j=1}^{M.M} x_j$) são normalmente utilizados como operações para diminuir as amostragens. Sendo M o tamanho da região de *pooling* e cada elemento da região de *pooling* representado através de $x_j = (x_1, x_2, \dots, x_{M.M})$, a saída após a passagem pela camada de *pool* é dada por x_i. O método de *pooling* máximo escolhe a característica mais invariante contida em determinada região, permitindo com esta técnica obter informações de textura levando a uma possível convergência mais rápida. O método de *pooling* médio processa a média de todas as características em determinada região mantendo assim a informação caracterizadora [157].

O *pooling* em pirâmide espacial [158], *pooling* estocástico [159], *def-pooling* [160], múltipla ativação de *pooling* [161], e preservação detalhada de *pooling* [162] são algumas das diferentes técnicas amplamente utilizadas em arquiteturas CNN [163].

Camada de Ativação

Os mapas de características da camada convolutiva são alimentados através de funções de ativação não lineares [164], fazendo com que toda a rede neuronal se aproxime de quase qualquer função não linear [165]. Se uma rede neuronal apenas usasse ativações lineares só seria capaz de realizar aproximações também estas lineares. A adição de mais camadas não melhoraria a sua expressividade. As funções de ativação são geralmente as unidades lineares com um propósito retificador muito simples, ou as ReLU's definidas como $ReLU(z) = \max(0, z)$, ou variantes desta, como as *Leaky* ReLU's ou as ReLU paramétricas [112]. No capítulo 2.4.5 pode ser encontrada informação mais detalhada sobre funções de ativação. A alimentação dos mapas de características através de uma função de ativação produz novos vetores, tipicamente também estes denominados por mapas de características.

Camadas Fully Connected (FC)

Na última camada da CNN é utilizada uma camada totalmente ligada (FC) tal como verificado na Figura 17. O princípio de funcionamento da FC na última camada da CNN é igual ao de uma rede neuronal tradicional. A camada FC recebe da camada de *pooling* um vetor e produz um vetor com N (número de classes) dimensões. Após passagem na camada de *pooling*, a característica dos mapas das camadas anteriores é comprimida e ligada às camadas FC.

Regularização de *Dropout's*

A introdução do mecanismo de regularização de desistências consiste numa ideia simples, mas com um grande impacto no desempenho das CNN's. Ao calcular a média de vários modelos de um conjunto, é normalmente conseguido um melhor desempenho do que quando se utilizam modelos únicos. O *Dropout* é um mecanismo para calcular a média baseada na amostragem estocástica das redes neuronais, prevenindo sobreajustamentos do modelo [166]. A ideia de *dropout* é utilizada noutros modelos de ML e DL da mesma forma que por exemplo a técnica de DART é utilizada árvores de decisão [167]. Paralelamente, ao remover aleatoriamente os neurónios regularizados durante o treino, consequentemente são utilizadas redes ligeiramente diferentes em cada lote, *batch*, dos dados de treino, sendo também os pesos da rede treinada, ajustados com base na otimização de múltiplas variações da rede.

Normalização dos *Batch*

As camadas de normalização dos lotes (*batch*) são tipicamente posicionadas depois das camadas de ativação, de forma a produzir esquemas de ativação normalizados, subtraindo a média e dividindo pelo desvio padrão em cada lote do conjunto de dados de treino. A implementação de camadas de normalização de lotes obriga a rede a alterar periodicamente as suas ativações para médias zero e desvio padrão da unidade, à medida que os lotes de treino atingem estas camadas. Este processo funciona como um regularizador da rede neuronal, acelerando o processo de treino, tornando-a menos dependente de uma iniciação criteriosa dos parâmetros da rede [134].

Modelos baseados nas CNN

Alguns modelos, com as respetivas configurações, baseadas na arquitetura CNN retratadas na literatura são apresentadas na Tabela 3.

Tabela 3. Arquiteturas baseadas nas CNN [168].

Modelo	Configuração	Referências
LeNet-5	2 camadas de convolução; 3 camadas totalmente ligadas (FC)	[169]
AlexNet	5 camadas de convolução; 3 camadas totalmente ligadas (FC)	[119]
VGG-16	13 camadas convolucionais; 3 camadas totalmente ligadas (FC)	[135]
Inception-v1	Arquitetura com 22 camadas; 5 M de parâmetros	[170]
Inception-v3	Inception-v3 é um predecessor da Inception-v1 com 24 M parâmetros	[15]
Inception-v4	Inception-v4 é constituída por duas partes: extrator de características e as camadas totalmente ligadas	[171]
Inception-ResNets	Possui 164 camadas de profundidade	[171]
ResNet-50	Arquitetura composta por 50 camadas	[132]
ResNeXt-50	Cinco módulos, cada módulo com um bloco de identificação e convolução. Cada bloco de convolução é constituído por 3 camadas e cada bloco de identidade possui 3 camadas de convolução.	[131]
Xception	36 camadas de convolução	[172]

Desde 1989 que têm sido efetuadas melhorias à arquitetura das CNN. Estas melhorias podem ser categorizadas como otimizações, parametrizações, regularização e até reformulações estruturais. Contudo, observa-se que o principal impulso nos resultados destas melhorias de desempenho das CNN, é oriundo das reestruturações efetuadas às arquiteturas com base nas CNN. Estas reestruturações contemplam melhorias nas unidades de processamento e da própria conceção dos blocos que constituem as CNN [173]. Dependendo do tipo de modificações efetuadas à arquitetura, as CNN podem ser amplamente categorizadas em diferentes classes (Figura 20).

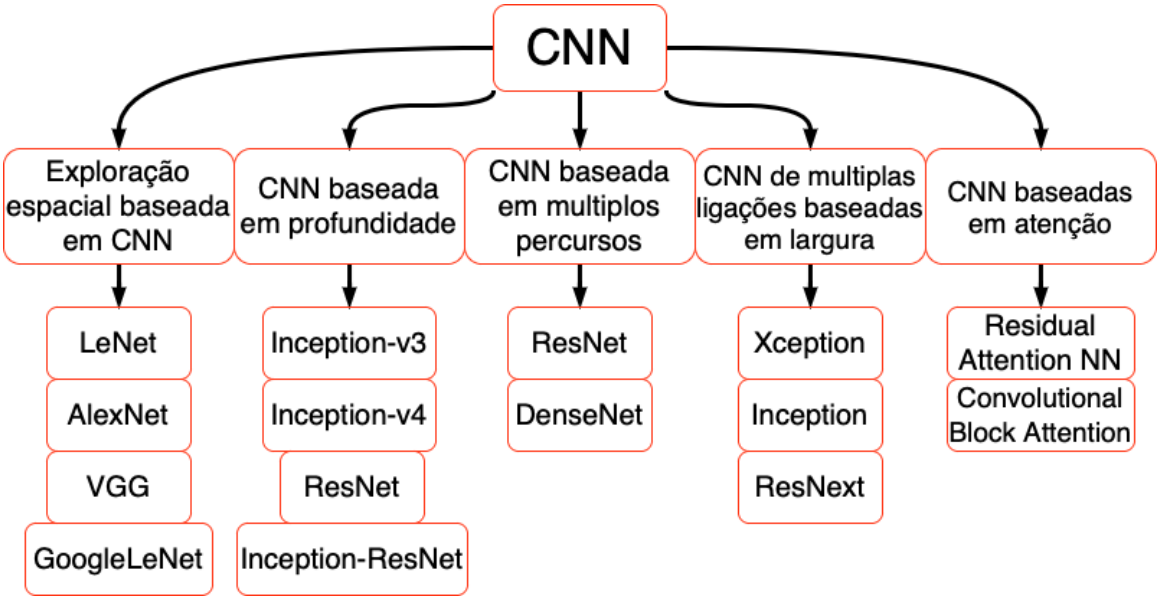


Figura 20. Categorias de arquiteturas baseadas nas CNN.

LeNet-5

A arquitetura *LeNet*-5, baseadas nas CNN, é composta por sete camadas e foi desenvolvida por Yann LeCunn (1990) para ser usada em reconhecimento de dígitos escritos à mão [174].

AlexNet

Krizhevsky *et al.* (2012) propuseram a arquitetura conhecida por *AlexNet*, onde esta consiste em uma *deep neural network* convolutiva composta por cinco camadas de convolução e três camadas FC. A arquitetura *AlexNet* de forma a tornar o processo de treino mais rápido, substitui a função de ativação *sigmoid* pela ReLU [119].

VGG

Simonyan *et al.* (2015) criaram os modelos VGG-16, também baseados na arquitetura das CNN, onde esta possui 13 camadas convolutivas e 3 camadas FC. O grupo de investigação *Visual Geometric Group* (VGG) desenvolveu juntamente com a VGG-16, os modelos VGG-11, VGG-13 e a VGG-19. O foco de estudo do grupo de investigação VGG é compreender como a profundidade das redes convolutivas afeta a precisão dos modelos de classificação e reconhecimentos de imagem [135]. De forma comparativa, o modelo VGG-19 possui 16 camadas convolutivas e 3 camadas FC, enquanto a VGG-11 possui apenas 8 camadas convolutivas e 3 camadas FC. Em todas as variações da arquitetura VGG são usadas 3 camadas FC, variando o número de camadas convolutivas [135].

GoogLeNet

Szegedy *et al.* (2015) propuseram uma rede para aplicações de classificação de imagens composta por 22 camadas diferentes denominada por *GoogLeNet* [170]. A ideia principal usada na implementação deste modelo foi a introdução de camadas *inception* onde estas submetem as camadas de entrada a operações de convolução utilizando filtros com diferentes tamanhos. A principal característica deste modelo consiste em uma melhor gestão dos recursos no interior da rede através do aumento da largura e profundidade da arquitetura [170].

ResNet

A arquitetura proposta por Kaiming *et al.* (2016) denominada por *ResNet* possui 33 camadas convolutivas e no final da rede apenas uma camada FC. Durante o desenvolvimento de redes neuronais ao longo dos anos, foram muitos os modelos que introduziram o princípio da utilização de múltiplas camadas ocultas nas suas arquiteturas. No entanto, a utilização de múltiplas camadas ocultas detona nos modelos problemas como o *vanishing* e o *exploding* gradiente. De forma a eliminar ou minimizar os problemas referidos em relação aos gradientes foram introduzidas as chamadas camadas *skip* para servirem de atalhos entre as camadas [132].

DenseNet

A *DenseNet* desenvolvida por Gao *et al.* (2017) consiste em uma arquitetura composta por vários blocos densos e de transição, que são dispostos entre dois blocos também densos, mas dispostos adjacentemente (Figura 21). Cada um dos blocos densos representa três camadas de normalização de lotes, *batch's,* seguidas de uma ReLU e uma operação de convolução com dimensão [3x3]

(Figura 21 (b)). Já os blocos de transição, são compostos por mecanismos de normalização dos *batch's*, convolução desta vez com dimensão [1x1] e uma camada de *pooling* médio [175].

Tradicionalmente, as redes neuronais do tipo *feed-forward* ligam a saída de uma camada à camada seguinte após aplicação de um conjunto de processos/operações. Estas operações incluem normalmente convolução, *pooling*, normalizações dos lotes e uma função de ativação através da equação $X_i = H_i(x_i - 1)$. A principal diferença da arquitetura *DenseNet* quando comparada com uma *ResNet* caracteriza-se pelo facto das *DenseNet* não somarem, mas sim concatenarem os mapas de características. Esta característica é expressa da seguinte forma: $X_i = H_i([x_0, x_1, \dots, x_{i-1}])$ [175].

A taxa de crescimento k, dada a concatenação dos mapas de características, é parte pertinente no grupo de hiper-parâmetros das ***DenseNet's***, e é expressa por $k_i = k_0 + k \,.\, (i - 1)$. Se cada função H_i produz k mapas de características, assume-se que a camada i^{th} possui $k_0 + k \,.\, (i - 1)$ mapas de características de entrada, onde k_0 corresponde ao número de canais existentes na camada de entrada [175].

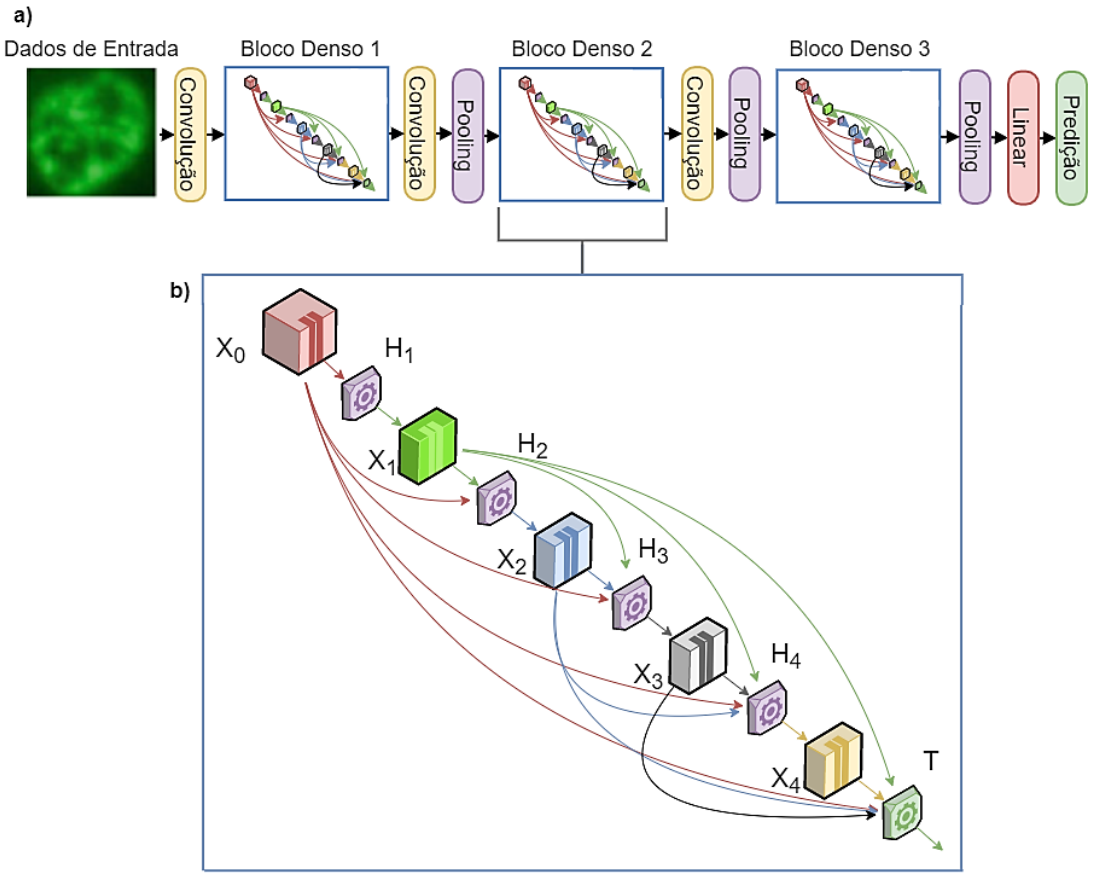


Figura 21. Arquitetura *DenseNet* com três blocos densos. **a**) arquitetura *DenseNet*; **b**) bloco denso.

No exemplo da Figura 21 (a), as camadas entre os dois blocos adjacentes são as denominadas camadas de transição com o propósito de mudarem o tamanho do mapa de características com recurso às operações de convolução e de *pooling* [175].

As arquiteturas *DenseNet* incluem variantes como por exemplo as *DenseNet-121* e a *DenseNet*-169 onde o número presente na denominação representa a "profundidade" da rede. A *DenseNet-121*, por exemplo, possui quatro blocos densos. Contemplando 6 camadas no bloco denso (BD) 1, 12 na BD2, 24 na BD3 e 16 na BD4. A $DenseNe121 = (5 + (6 + 12 + 24 + 16).2)$, onde 5 representa uma camada de convolução inicial, uma de *pooling* e mais 3 de transição, enquanto a multiplicação por 2 em cada BD diz respeito a uma camada de convolução e uma de *pooling* presente em cada bloco denso.

CapsNet

Capsule Networks (*CapsNet*) foram desenvolvidas na tentativa de superar a perda de informação que por vezes ocorre com arquiteturas como a CNN durante os processos e a passagem pelas camadas de *pooling*. As *CapsNet* são compostas por um tipo especial de neurónio denominado por capsulas, onde estas conseguem detetar eficazmente informação distinta entre amostras, informação espacial assim como as características mais significativas [176]. Este modelo constituído por cápsulas é composto por quatro componentes/processos relevantes, nomeadamente a, multiplicação matricial, ponderação escalar dos valores de entrada, algoritmo de encaminhamento dinâmico e a função de compressão.

EfficientNet

A *EfficientNet* (EN) é uma arquitetura baseada nas CNN com a adição de esta integrar metodologias de redimensionamento uniforme de dimensões de profundidade, largura e resolução através de um coeficiente composto com escalas fixas. A variante *EfficientNet-B0* é composta por blocos de compressão e excitação e por um bloco residual de bloqueio invertido baseado nos blocos encontrados na arquitetura da *MobileNetV2*.

2.4.6.5 Redes Neuronais Recorrentes

As redes neuronais recorrentes (RNN) fazem parte de uma classe de redes neuronais utilizadas para processar informação sequencial. A estrutura das RNN, tal como ilustrado na Figura 22, possui uma arquitetura semelhante às FFNN com a diferença nas ligações recorrentes que são introduzidas nos nós das camadas ocultas (*Context Nodes*) [177].

As duas abordagens de RNN mais utilizadas dizem respeito às tipologias de *Elman* (Figura 22) e à de *Jordan.* No caso da rede de *Elman*, a camada oculta alimenta uma camada de estados nos nós de contexto, onde estas retêm a memória de entradas já passadas. A tipologia de *Jordan*, ao contrário da abordagem anterior, em vez de manter o histórico da camada oculta, armazena a informação da camada de saída na camada de estado.

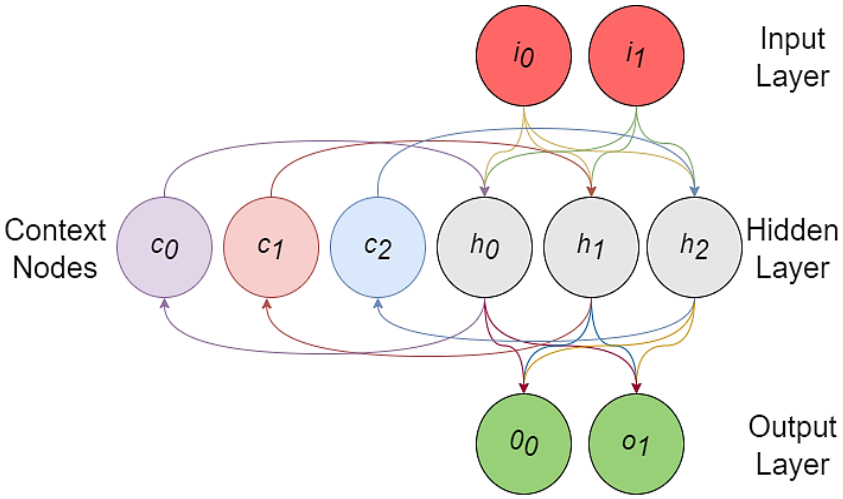


Figura 22. Rede Neuronal Recorrente (RNN) - Tipologia de *Elman.*

Genericamente, uma unidade de ligação recorrente oculta de uma RNN h_t, em determinado valor de t, recebe a ativação de entrada de dados atuais i_t assim como do estado anterior oculto h_{t-1}. A saída o_t (y) é calculada de acordo com o estado oculto em h_t. A expressão matemática que descreve as RNN é dada por $h_t = f(w_{hx}x_t + w_{hh}h_{t-1} + b_h)$ e $y = softmax(w_{yh} + b_y)$. A função de ativação não linear é representada por f, a matriz dos correspondentes pesos entre a camada de entrada e a camada oculta caracterizasse por w_{hx}, w_{yh} diz respeito à matriz de pesos entre a camada oculta e a camada de saída e por fim, b_h representa as *biases* [178].

Apesar da arquitetura das RNN se caracterizar de uma forma simples e eficiente, é infelizmente, difícil de treinar adequadamente. Algoritmos como o *Real-Time Recurrent Learning* (RTRL) [179] e o *Back Propagation Through Time* (BPTT) [180] são muitas vezes utilizados para treinar as RNN. No entanto o processo de aprendizagem com recurso aos métodos referidos falha frequentemente devido à problemática associada ao *vanishing gradient*, que ocorre durante a multiplicação de muitos valores pequenos, ou à *explode gradient* que neste caso ocorre na multiplicação de muitos valores com grande dimensão [181, 182]. Hochreiter *et al* (1997) desenvolveram um novo modelo baseado nas RNN denominado por *Long Short Term Memory* (LSTM) que supera os problemas associadas ao refluxo de erros com a ajuda de uma célula de memória especialmente concebida [183]. A célula de memória usada nas LSTM é tipicamente configurada com três portas: porta de entrada (g_t), porta de esquecimento (f_t) e a porta de saída (o_t). As portas utilizadas no módulo de memória permitem adicionar ou remover informação da mesma.

2.4.6.6 Redes Generativas Adversárias

As *Generative Adversial Networks* (GAN) fazem parte do grupo de modelos generativos em DL, sendo esta, introduzida por Fellow *et al.* (2014) [184] (Figura 23). As GAN fazem parte de um grupo de redes neuronais com a capacidade de gerar imagens sintéticas, de forma a imitarem as imagens originais utilizadas no processo de treino.

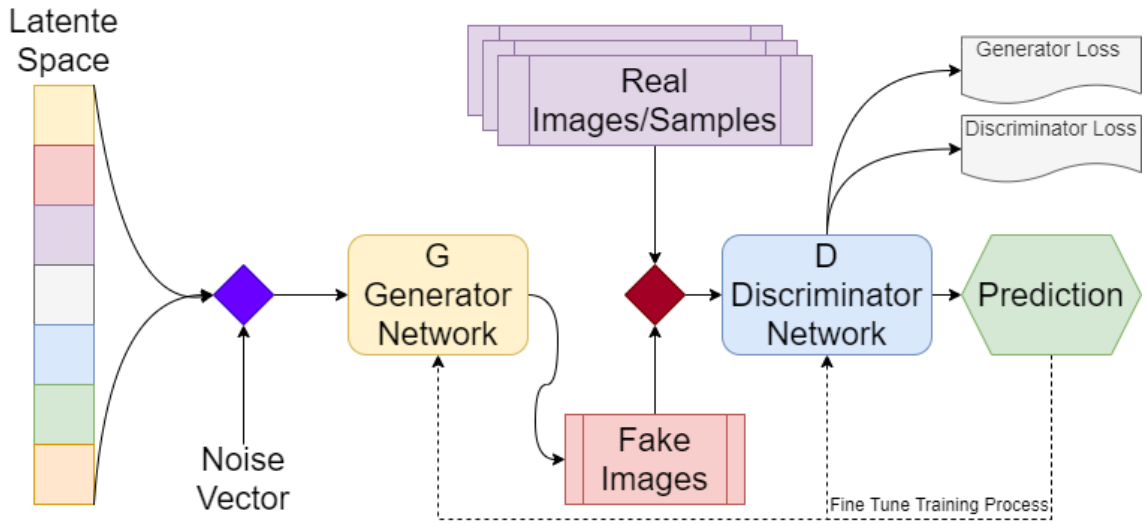


Figura 23. Generative Adversial Networks

A arquitetura GAN da Figura 23 é composta por duas redes neuronais, a geradora (G) e a discriminadora (D), que passam ambas simultaneamente pelo processo de aprendizagem. O gerador gera amostras de dados falsos, enquanto o discriminador tenta distinguir corretamente as amostras verdadeiras das amostras falsas. Matematicamente, as redes neuronais D e G, disputam entre si uma espécie de "jogo" de *minimax* com recurso à função de custo expressa da seguinte forma: $min_G max_D V(D,G) = E_{x\sim p_{data(x)}}[\log D\,(x)] + E_{z\sim p_{z(z)}}\left[\log(1 - D\big(G(z)\big))\right]$ [185]. A imagem original é expressa em x, z compreende um vetor de ruído adicionado aleatoriamente ao modelo. $p_{data}(x)$ e p_z são distribuições de probabilidades de x e de z, respetivamente. $D(x)$ representa também uma probabilidade, neste caso, de que x é oriundo dos dados reais $p_{data}(x)$ e não dos dados gerados pela rede geradora G. A probabilidade do que pode ser gerado a partir de $p_z(z)$ é representado por $1 - D(G(z))$. No caso da expectativa de x a partir da distribuição dos dados reais p_{data}, esta é expressa por $E_{x\sim p_{data}(x)}$, enquanto a expetativa da amostra z oriunda do ruído introduzido é expressa por $E_{z\sim p_{z(z)}}$. O objetivo do processo de aprendizagem nas GAN consiste em maximizar a função de perda para o discriminador, enquanto o objectivo da aprendizagem do gerador é reduzir o termo $\log(1 - D\big(G(z)\big))$ [185].

A arquitetura GAN é amplamente utilizada no campo de estudo de análise de imagiologia médica, com foco na capacidade de estas conseguirem gerar novos dados (capacidade de gerar imagens sintéticas) assim como nos mecanismos de tradução imagem-imagem [186]. No entanto, apesar da sua utilização, a confiança e avaliação dos dados gerados, e o instável processo de treino são alguns dos motivos que dificultam a sua aceitação por parte da comunidade médica [187].

2.4.6.7 Arquiteturas de Segmentação

O processo de divisão de uma imagem em diferentes regiões onde a informação necessária para um processamento posterior possa ser extraída é conhecida por segmentação. Este processo é basicamente a separação de uma região de interesse (ROI) do fundo da própria imagem. ROI é a parte da imagem que queremos utilizar, ou seja, a região de interesse que no caso de imagens

cancerosas, é necessário a região da lesão para extrair as características da parte doente. A segmentação pode ser dividida em quatro classes principais:

i) Segmentação baseada em limiares (*Threshold-based*);
ii) Segmentação com base na região;
iii) Segmentação baseada em pixéis;
iv) Segmentação baseada em modelos;

A segmentação baseada em lineares inclui o método de *Otsu*, entropia máxima, limiar local e global e limiar baseado em histogramas [188]. Os métodos de segmentação *Watershed-based* e projeções de múltiplas resoluções são alguns dos exemplos de segmentação baseada na região [189]. Agrupamentos de *Fuzzy c-means*, redes neuronais artificiais e o método de *Markov* são alguns dos métodos da classe de segmentação baseada em pixéis. Quanto à segmentação baseada em modelos, este consiste na utilização de um modelo paramétrico deformável como o caso dos conjuntos de níveis. Existem muitos outros métodos de segmentação de imagens: limiar de histogramas, limiar adaptativo [190], vetores de fluxo de gradiente, identificação de região distribuída e localizada, agrupamentos e regiões estatísticas [191], aprendizagem *bootstrap* [192], contornos ativos, aprendizagem supervisionada, deteção de limites e arestas, modelação probabilística, codificação dispersa [193] e segmentação cooperativa de redes neuronais [194]. Modelos híbridos destes métodos combinando duas ou mais metodologias foram utilizados para melhorar precisão do sistema [195].

U-NET

Ronneberger *et al.* (2015) propõem uma arquitetura *U-NET* baseada nas CNN para ser utilizada em aplicações de segmentação de dados de imagens biomédicas [196]. Este modelo é utilizado em tarefas de quantificação, tais como, na deteção de células e na medição da geometria de dados em imagens médicas [197]. A arquitetura é composta por dois caminhos, um de contração para capturar o contexto dos dados situada do lado esquerdo, e o caminho expansivo simétrico, do lado direito, com o propósito de com um elevado grau de precisão, localizar os dados (Figura 24) [196]. A estratégia usada no processo de aprendizagem das *U-NET* depende da utilização de *augmentation* para um treino eficaz a partir de poucas amostras acompanhadas de anotação [196]. A *U-NET*, arquitetura em forma de U, consiste num esquema específico de codificador-descodificador: O codificador reduz as dimensões espaciais em cada uma das camadas aumentando ao mesmo tempo os canais. Por outro lado, a componente descodificadora, aumenta as dimensões espaciais ao mesmo tempo que reduz os canais. No final, as dimensões espaciais são restauradas para fazer uma previsão para cada *pixel* presente na imagem de entrada (Figura 24).

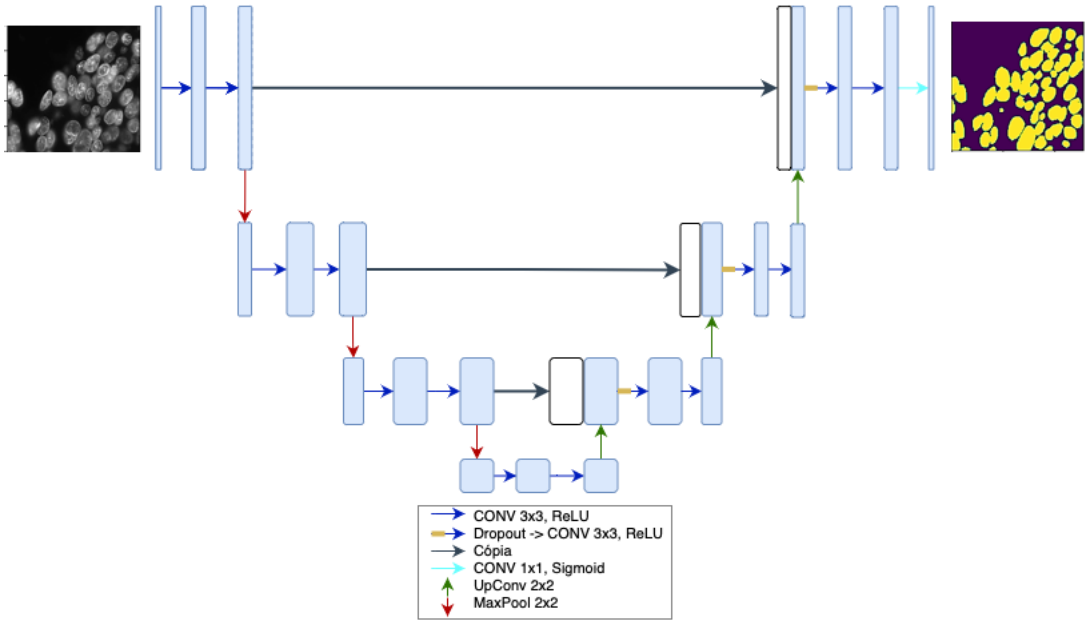


Figura 24. Arquitetura *U-NET.*

Foram desenvolvidas várias extensões da *U-NET* para diferentes tipos de imagens e domínios com requisitos específicos. Por exemplo, Zhou *et al.* (2018) desenvolveram uma arquitetura *U-NET* "aninhada", também conhecida por *U-NET*++, para segmentação de imagens médicas [198]; Zhang *et al.* (2018) desenvolveram um algoritmo de segmentação baseado nas *U*-NET, usada em imagens de estradas [199]; e Çiçek *et al.* (2016) propuseram uma arquitetura *U-NET* para imagiologia 3D [200].

Capítulo 3: Metodologias e Desenvolvimento

3.1 Ferramentas de Suporte

No desenvolvimento desta dissertação foram utilizadas diversas ferramentas de forma a concretizar os objetivos propostos, i.e., linguagem de programação, bibliotecas e hardware, que se descreve de seguida. A linguagem de programação utilizada para o desenvolvimento dos algoritmos, preparação e o tratamento dos dados foi o *Python* (versão 3.8) [201]. *Tensorflow* [202] e *PyTorch* [203] foram as duas bibliotecas de ML e DL utilizadas no desenvolvimento e implementação dos algoritmos propostos. *Numpy* [204] e *Pandas* [205] foram as bibliotecas utilizadas na preparação e manipulação dos dados, enquanto as bibliotecas *Seaborn* [206] e *Matplotlib* [207] foram usadas para a visualização de dados assim como dos resultados. *Scikit-learn* [208] foi utilizada na aplicação de algumas métricas para a classificação dos modelos. *Streamlit* é uma biblioteca para a criação e distribuição de aplicações de *machine learning* e de análise de dados e foi utilizada para o desenvolvimento de uma plataforma de demonstração de resultados [209]. O IDE utilizado para o desenvolvimento de software foi o da plataforma da *JetBrains* [210] denominada por *PyCharm* [211].

As características de *hardware* da máquina utilizada no desenvolvimento do trabalho desta dissertação estão presentes na Tabela 4.

Tabela 4. Configurações da máquina utilizada para o desenvolvimento do trabalho.

	Máquina
Sistema Operativo	Windows 10 Pro (21H1 - 19043.1023)
CPU	i7-7700HQ 2.8 GHz
GPU	GTX 1060m 6 GB (461.40)
RAM	32 GB

3.2 Etapas do Desenvolvimento

Para cada uma das tarefas foi selecionado um conjunto de dados. Após esta seleção, foi criado um processo automatizado, com várias etapas, para cada uma das tarefas deste trabalho (Figura 25). Assim, numa fase inicial do processo, os dados foram lidos e submetidos a técnicas de pré-processamento, como por exemplo o redimensionamento de imagem. Em seguida, o conjunto de dados foi subdividido em três componentes e submetidos às técnicas de processamento definidas. Posteriormente, para cada um dos modelos implementados em cada uma das tarefas, foi iniciado o processo de treino, parametrização e otimização. Por fim, o processo de treino anteriormente efetuado é testado, avaliado e analisado.

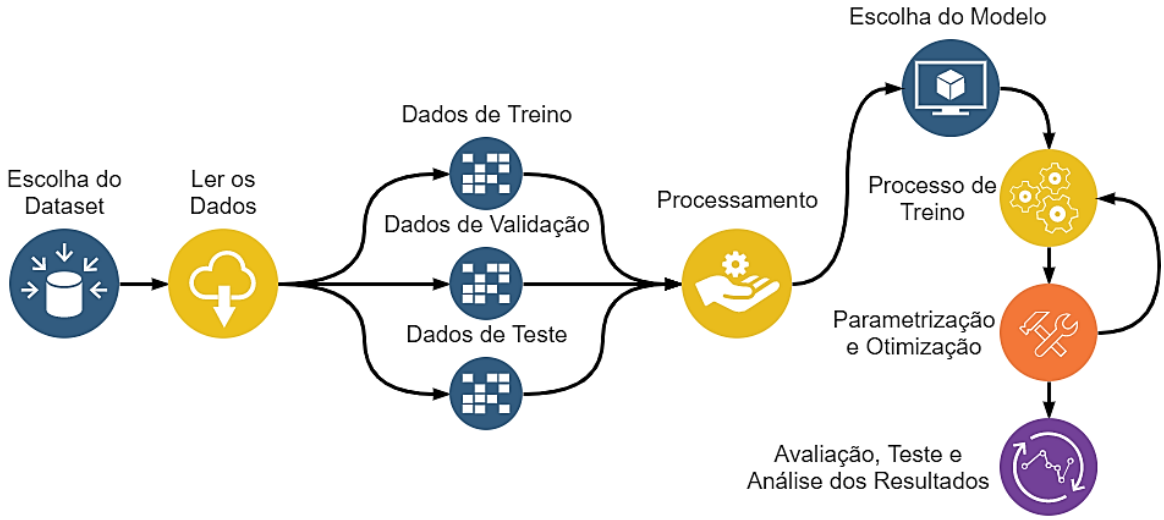


Figura 25. Etapas tidas em consideração no desenvolvimento deste trabalho.

3.3 Classificação de Imagens

Datasets

Os conjuntos de dados utilizados para a implementação de metodologias de classificação de imagens médicas foram o dataset denominado por *MedMNIST* [212] e *MedNIST* [213].

O *MedMNIST* corresponde a uma coleção de 10 conjuntos de imagens médicas, com dimensionamento original de 28x28 pixels. Os dados incluídos no *MedMNIST* incluem imagens oriundas de estudos de histologia (PathMNIST) [214], raios-x à mama (ChestMNIST) [215], imagens obtidas em estudos dermatológicos (DermaMNIST) [216], estudos oculares com imagens obtidas em tomografias (OCTMNIST) [217], raios-x ao tórax em estudos pneumológicos (PneumoniaMNIST) [217], estudos na patologia de diabetes com recurso a imagens de retina (RetinaMNIST) [218], ultrassom à mama no estudo de cancros (BreastMNIST) [219], imagens 3D de tomografias ao fígado originalmente utilizado para deteção de órgãos divididas nos subconjuntos OrganMNISTAxial, OrganMNISTCoronal e OrganMNISTSagittal [220].

Deste modo, o *MedMNIST* é um dataset composto por 10 classes de dados médicos pré-processados, padronizado, para realizar tarefas de classificação. As classes incluem modalidades de estudo médico desde a patologia, dermatologia, ultrassons, tomografias e raios-x.

Na Tabela 5 está contida a informação sobre o número de amostras, as subclasses presentes em cada classe do conjunto de dados *MedMNIST,* assim como, as tarefas de classificação.

Tabela 5. Classes, subclasses e respetiva distribuição de amostras no *MedMNIST.*

Classe	Subclasses	Nº de amostras	Tarefas
PathMNIST	0: Adipose; 1: Background; 2: Debris; 3: Lymphocytes; 4: Mucus; 5: Smooth Muscle; 6: Normal Colon Mucosa; 7: Cancer-associated Stroma; 8: Colorectal Adenocarcinoma Epithelium.	107180	- Múltiplas classes
ChestMNIST	0: Atelectasis; 1: Cardiomegaly; 2: Effusion; 3: Iinfiltration; 4: Mass; 5: Nodule; 6: Pneumonia; 7: Pneumothorax; 8: Consolidation; 9: Edema; 10: Emphysema; 11: Fibrosis; 12: Pleural; 13: Hernia.	112120	- Múltiplas Etiquetas - Classe Binária
DermaMNIST	0: Actinic Keratoses and Intraepithelial Carcinoma; 1: Basal Cell Carcinoma; 2: Benign Keratosis-like lesions; 3: Dermatofibroma; 4: Melanoma; 5: Melanocytic Nevi; 6: Vascular lesions.	10015	- Múltiplas classes
OCTMNIST	0: Choroidal Neovascularization; 1: Diabetic Macular Edema; 2: Drusen; 3: Normal.	109309	- Múltiplas classes
PneumoniaMNIST	0: Normal; 1: Pneumonia.	5856	- Classe Binária
RetinaMNIST	5 niveis de serveridade da retinopatia diabética.	1600	- Regressão Ordinal
BreastMNIST	0: Malignant; 1: Normal; 2: Benign.	780	- Classe Binária
OrganMNISTAxial	0: Bladder; 1: Femur-left; 2: Femur-right; 3: Heart; 4: Kidney-right; 5: Kidney-left; 6: Liver; 7: Lung-left; 8: Lung-right; 9: Pancreas; 10: Spleen.	58850	- Múltiplas classes
OrganMNISTCoronal	0: Bladder; 1: Femur-left; 2: Femur-right; 3: Heart; 4: Kidney-right; 5: Kidney-left; 6: Liver; 7: Lung-left; 8: Lung-right; 9: Pancreas; 10: Spleen.	23660	- Múltiplas classes

OrganMNISTSagittal	0: Bladder; 1: Femur-left; 2: Femur-right; 3: Heart; 4: Kidney-right; 5: Kidney-left; 6: Liver; 7: Lung-left; 8: Lung-right; 9: Pancreas; 10: Spleen.	25221	- Múltiplas classes

O conjunto de dados *MedNIST* está dividido em seis classes com imagens obtidas a partir de vários conjuntos da TCIA [221], RSNA *Bone Age Challenge* [222], e do conjunto de dados de raios-x ao tórax do NIH [215]. As classes do conjunto de dados incluem imagens de tomografias ao tórax (CXR), abdómen (*AbdomenCT*), mama (*ChestCT*) [215] e cabeça (*HeadCT*), ressonâncias magnéticas da mama (*BreastMRI*) e radiografias de mãos (*Hand*). O tamanho das imagens do *MedNIST* foi redimensionado pelos autores para 64x64 *pixels* [213].

Na Tabela 6 estão presentes as distribuições do número de amostras por classe do conjunto de dados *MedNIST.*

Tabela 6. Distribuição de amostras no *MedNIST.*

Classe	AbdomenCT	BreastMRI	CXR	ChestCT	Hand	HeadCT
Nº de amostras	10000	8954	10000	10000	10000	10000

Processamento dos Datasets

- **MedMNIST**

O processamento efetuado ao conjunto de dados MedMNIST divide-se em quatro fases de transformações que foram efetuadas no decorrer do estudo efetuado de forma a perceber o impacto destas nos resultados obtidos (Tabela 7). O primeiro conjunto de transformações de processamento ao dataset inclui normalização de imagem com parâmetros específicos de média de 0,5 e desvio de 1,0 (saída da normalização = entrada – média / desvio) e a conversão da imagem (*array*) para tensor (*array* multidimensional imutável). A terceira configuração de transformações inclui o mesmo processamento anteriormente discutido, mas desta vez com o parâmetro de média da normalização especificado em 1,0. Em uma tentativa de perceber o impacto do pré-processamento, foi também criada a configuração 2 onde a única transformação efetuada foi a conversão da imagem em tensor. Por último, a quarta configuração, inclui a normalização de imagem (média de 0,5 e desvio de 1,0), conversão de imagem em tensor, dimensionamento da intensidade da imagem com fator aleatório (imagem = imagem * (1 + fator)) e a aplicação de rotações aleatórias às imagens.

- **MedNIST**

As configurações de processamento efetuadas ao conjunto de dados MedNIST incluem normalização de imagem, rotações e inversões aleatórias às imagens. As imagens originalmente são fornecidas com dimensão de 64x64 e foi mantida esta dimensão de imagem. Incluído no processamento é ainda convertida a imagem (matriz) em um tensor (*array* multidimensional imutável). Na Figura 26 encontram-se alguns exemplos das transformações e processamento efetuado ao dataset MedNIST.

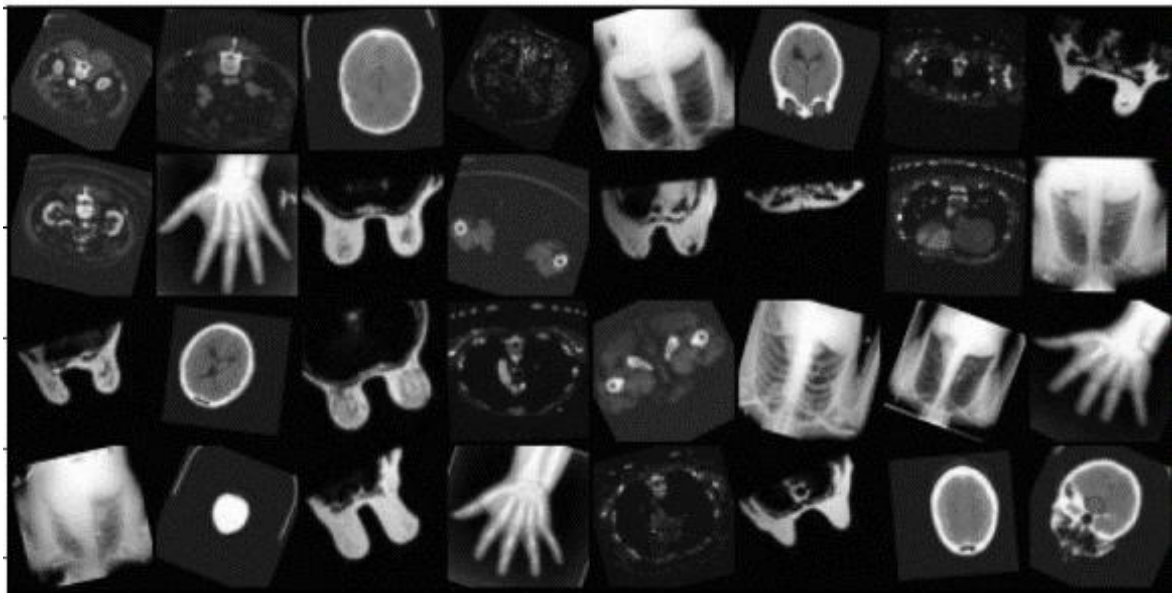


Figura 26. Exemplos de imagens do dataset MedNIST depois de transformadas.

3.4 Arquiteturas de Classificação de Imagem

De seguida serão apresentadas as arquiteturas contruídas e implementadas nas tarefas de classificação de imagens dos conjuntos de dados *MedMNIST e MedNIST.* Os modelos implementados incluem a *CNN*, *DenseNet, ResNet, VGG16* e a *EfficientNet-B0.*

3.4.1 CNN

A primeira arquitetura desenvolvida, tanto para as tarefas de classificação no dataset *MedMNIST* como no *MedNIST*, foi uma CNN. A construção da CNN usada no *MedMNIST* (Figura 27) foi realizada com recurso a *PyTorch,* mais concretamente com a ferramenta de contentores sequenciais, *sequential*, onde é possível adicionar módulos, como as funções de ativação e as operações de *pooling*, e definir a arquitetura da rede. No caso da CNN usada no *MedNIST* (Figura 28), esta também foi construída com recurso a *PyTorch*, mas neste caso, usando a classe *functional*.

A CNN (subcapítulo 2.4.6.4) desenvolvida para o *MedMNIST* é composta por seis camadas, sendo a última uma camada totalmente ligada (FC) (Figura 27). Todas as camadas, com exceção da FC,

integram operações de convolução 2D (*Conv2d*) e operações de normalização nos lotes (*batch*) (*BatchNorm2d*). Nas camadas dois e cinco foram também implementadas as operações de *pooling*, nomeadamente, a variante de *pooling* máximo (*MaxPool2d*). Na camada FC, à semelhança das restantes camadas, foi utilizada a função de ativação ReLU (Tabela 1, No.5), tendo sido intercaladas operações de transformação linear entre as entradas e saídas das camadas de ativação. A função de otimização usada na CNN, desenvolvida para as tarefas de classificação no *MedMNIST,* foi a descida de gradiente estocástica (SGD) juntamente com a implementação do parâmetro *momentum* [223]. Quanto à função de calculo das perdas, foi automatizado o processo de escolha desta função, de forma a esta variar consoante o tipo de tarefa incluído no dataset *MedMNIST* (Tabela 5). Em tarefas de múltiplas etiquetas ou em classes/classificação binária é usada a função *BCEWithLogitsLoss,* presente na oferta de ferramentas do *PyTorch*, que consiste em uma combinação de uma camada *Sigmoid* com a função de cálculo de perdas de entropia binária cruzada (BCE). Nas restantes tarefas presentes no dataset *MedMNIST* foi utilizada a função *CrossEntropyLoss* [224]. As métricas de desempenho utilizadas foram a exatidão (ACC) e a área sob a curca ROC (AUC) (subcapítulo 2.4.1).

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
CNN                                      --                        --
├─Sequential: 1-1                        [128, 16, 26, 26]         --
│    └─Conv2d: 2-1                       [128, 16, 26, 26]         448
│    └─BatchNorm2d: 2-2                  [128, 16, 26, 26]         32
│    └─ReLU: 2-3                         [128, 16, 26, 26]         --
├─Sequential: 1-2                        [128, 16, 12, 12]         --
│    └─Conv2d: 2-4                       [128, 16, 24, 24]         2,320
│    └─BatchNorm2d: 2-5                  [128, 16, 24, 24]         32
│    └─ReLU: 2-6                         [128, 16, 24, 24]         --
│    └─MaxPool2d: 2-7                    [128, 16, 12, 12]         --
├─Sequential: 1-3                        [128, 64, 10, 10]         --
│    └─Conv2d: 2-8                       [128, 64, 10, 10]         9,280
│    └─BatchNorm2d: 2-9                  [128, 64, 10, 10]         128
│    └─ReLU: 2-10                        [128, 64, 10, 10]         --
├─Sequential: 1-4                        [128, 64, 8, 8]           --
│    └─Conv2d: 2-11                      [128, 64, 8, 8]           36,928
│    └─BatchNorm2d: 2-12                 [128, 64, 8, 8]           128
│    └─ReLU: 2-13                        [128, 64, 8, 8]           --
├─Sequential: 1-5                        [128, 64, 4, 4]           --
│    └─Conv2d: 2-14                      [128, 64, 8, 8]           36,928
│    └─BatchNorm2d: 2-15                 [128, 64, 8, 8]           128
│    └─ReLU: 2-16                        [128, 64, 8, 8]           --
│    └─MaxPool2d: 2-17                   [128, 64, 4, 4]           --
├─Sequential: 1-6                        [128, 9]                  --
│    └─Linear: 2-18                      [128, 128]                131,200
│    └─ReLU: 2-19                        [128, 128]                --
│    └─Linear: 2-20                      [128, 128]                16,512
│    └─ReLU: 2-21                        [128, 128]                --
│    └─Linear: 2-22                      [128, 9]                  1,161
==========================================================================
Total params: 235,225
Trainable params: 235,225
Non-trainable params: 0
Total mult-adds (M): 952.74
==========================================================================
Input size (MB): 1.20
Forward/backward pass size (MB): 71.18
Params size (MB): 0.94
Estimated Total Size (MB): 73.33
==========================================================================
```

Figura 27. Arquitetura CNN utilizada no *MedMNIST.*

No caso do modelo CNN usado no *MedNIST*, este é composto por cinco camadas, onde as três últimas camadas são do tipo FC e as duas primeiras são de convolução. Na construção da CNN (Figura 28) foram testadas duas funções de ativação, nomeadamente, a ReLU (Tabela 1, No. 5) e a ELU (Tabela 1, No. 7). No entanto, foi optado pela utilização da ELU de forma a diminuir a complexidade computacional e aumentar a velocidade do processo de treino. A ELU foi então utilizada entre todas as camadas com a exceção da última FC, de forma a ser possível calcular as perdas. Este modelo consiste em uma simples implementação de uma CNN, para averiguar a sua capacidade de aprendizagem, tempo usado no processo de treino, assim como, da sua complexidade computacional comparativamente aos restantes modelos desenvolvidos. A técnica utilizada para otimização, à semelhança da utilizada na CNN anterior, foi a SGD. A função aplicada para o cálculo das perdas foi a entropia cruzada (*CrossEntropyLoss*), enquanto a métrica de desempenho utilizada foi a exatidão (ACC).

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
CNN                                      --                        --
├─Conv2d: 1-1                            [300, 5, 58, 58]          250
├─Conv2d: 1-2                            [300, 10, 52, 52]         2,460
├─Linear: 1-3                            [300, 400]                10,816,400
├─Linear: 1-4                            [300, 80]                 32,080
├─Linear: 1-5                            [300, 6]                  486
==========================================================================
Total params: 10,851,676
Trainable params: 10,851,676
Non-trainable params: 0
Total mult-adds (G): 5.50
==========================================================================
Input size (MB): 4.92
Forward/backward pass size (MB): 106.43
Params size (MB): 43.41
Estimated Total Size (MB): 154.75
==========================================================================
```

Figura 28. Arquitetura CNN utilizada no *MedNIST.*

3.4.2 ResNet18

A arquitetura *ResNet18* implementada para o *MedMNIST* (Figura 29) contempla duas primeiras camadas, uma de convolução (*Conv2d*) e uma outra para a normalização de lotes (*BatchNorm2d*), e quatro módulos *sequential,* onde cada um destes possui dois módulos denominados por *BasicBlock*. Cada um dos *BasicBlock* é constituído por duas camadas de convolução, uma de normalização de lotes entre as de convolução e, por fim, uma camada de ligação ao próximo módulo. Após o último módulo *sequential* (Figura 29, *Sequential* 1-6) foi utilizada uma camada de *pooling* médio (*avg_pool2d*) e uma última camada FC linear. Quanto à função de ativação, foi utilizada a ReLU. As medidas de desempenho, funções de otimização e cálculo de perdas foram as mesmas utilizadas na CNN construída para o *MedMNIST* (Figura 27).

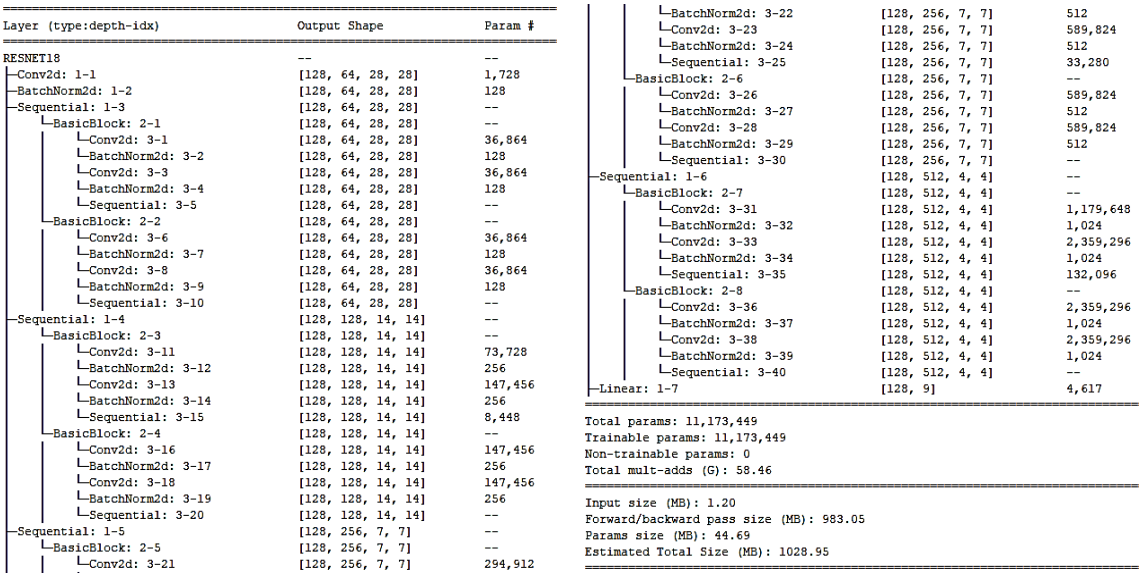


Figura 29. Arquitetura *ResNet18* utilizada no *MedMNIST.*

A *ResNet18* usada no *MedNIST* (Figura 30) à semelhança da anterior, possui quatro módulos *sequential,* onde cada um destes possui dois *BasicBlock*. Os *BasicBlock* implementados representam as operações residuais das *ResNet*. Esta *ResNet18* é composta por uma primeira camada de convolução (*Conv2d*), seguida de uma camada de normalização de lotes (*BatchNorm2d*) e de uma de camada de *pooling* na variante de *pooling* máximo. Cada um dos *BasicBlock* contêm duas camadas de convolução e duas de normalização de lotes intercaladas entre si. As duas últimas camadas correspondem a uma camada de *pooling* médio adaptativo (*AdaptiveAvgPool2d*) e a uma camada FC linear. Nesta *ResNet18* foi também aplicada a função ReLU nas camadas de ativação. A métrica utilizada foi a exatidão (ACC), a função de otimização implementada foi a SGD e a função de perdas foi a entropia cruzada (*CrossEntropyLoss*).

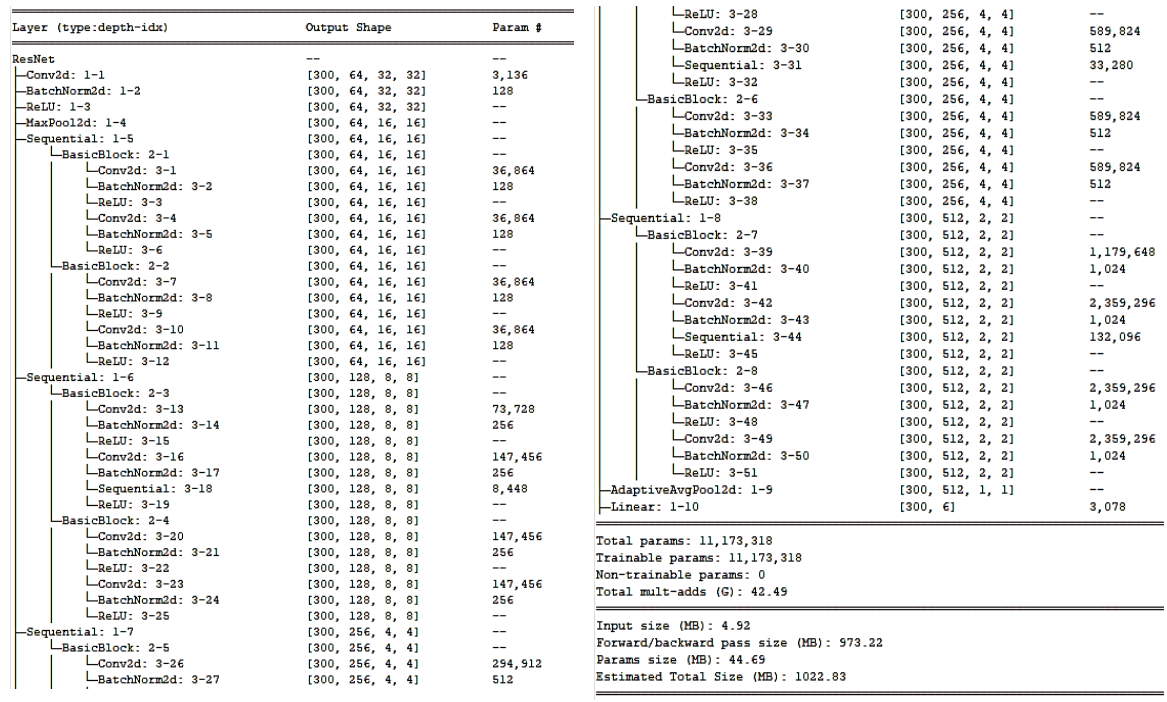
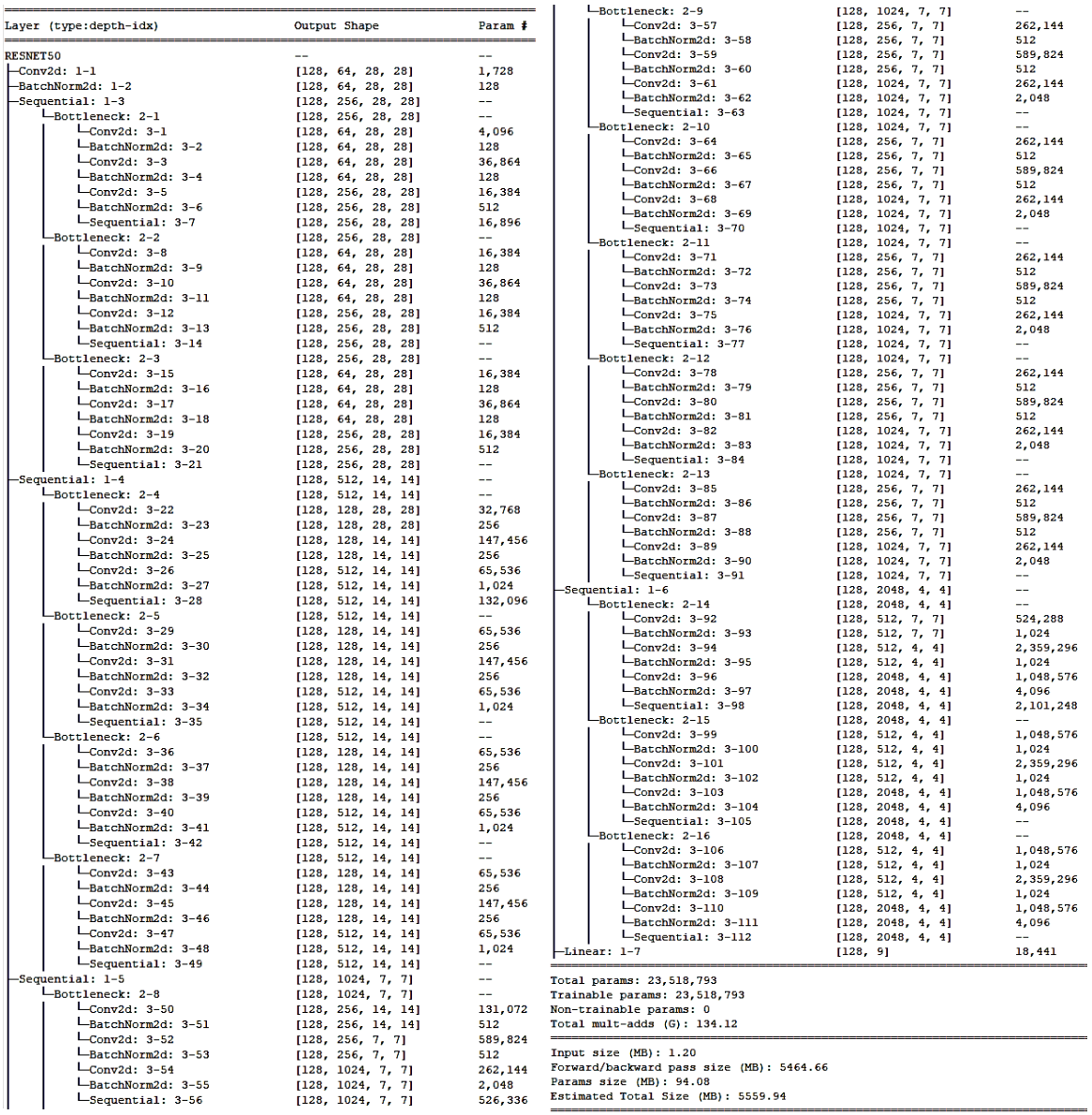


Figura 30. Arquitetura *ResNet18* utilizada no *MedNIST.*

3.4.3 ResNet50

A *ResNet50* implementada para o *MedMNIST* (Figura 31) foi também construída com recurso à técnica *sequential* e contempla blocos específicos de camadas neste caso denominados por *Bottleneck*. A utilização dos blocos *Bottleneck* permite aumentar a profundida da rede, mantendo, contudo, o tamanho dos parâmetros da rede o mais baixo possível [225]. A arquitetura foi construída com uma primeira camada de convolução, seguida de uma camada de normalização de lotes. Esta termina com uma camada de *pooling* médio e uma FC linear na saída. A rede contém quatro módulos *sequencial*, tendo os módulos 1-3 e 1-6 dois blocos *Bottleneck,* o módulo 1-4 quatro blocos *Bottleneck*, e o módulo 1-5 seis blocos *Bottleneck*. Por sua vez, cada um dos *Bottleneck* é composto por camadas de convolução, camadas de normalização de lotes intercaladas entre si, e pelas camadas de ativação. As camadas de ativação implementadas na *ResNet50* usam a função de ativação ReLU (Figura 31). À semelhança das implementações anteriores efetuadas para o conjunto de dados *MedMNIST*, as métricas de desempenho, funções de cálculo das perdas e a função de otimização, foram as mesmas.



Layer (type:depth-idx)	Output Shape	Param #
RESNET50	--	--
├─Conv2d: 1-1	[128, 64, 28, 28]	1,728
├─BatchNorm2d: 1-2	[128, 64, 28, 28]	128
├─Sequential: 1-3	[128, 256, 28, 28]	--
│ └─Bottleneck: 2-1	[128, 256, 28, 28]	--
│ │ └─Conv2d: 3-1	[128, 64, 28, 28]	4,096
│ │ └─BatchNorm2d: 3-2	[128, 64, 28, 28]	128
│ │ └─Conv2d: 3-3	[128, 64, 28, 28]	36,864
│ │ └─BatchNorm2d: 3-4	[128, 64, 28, 28]	128
│ │ └─Conv2d: 3-5	[128, 256, 28, 28]	16,384
│ │ └─BatchNorm2d: 3-6	[128, 256, 28, 28]	512
│ │ └─Sequential: 3-7	[128, 256, 28, 28]	16,896
│ └─Bottleneck: 2-2	[128, 256, 28, 28]	--
│ │ └─Conv2d: 3-8	[128, 64, 28, 28]	16,384
│ │ └─BatchNorm2d: 3-9	[128, 64, 28, 28]	128
│ │ └─Conv2d: 3-10	[128, 64, 28, 28]	36,864
│ │ └─BatchNorm2d: 3-11	[128, 64, 28, 28]	128
│ │ └─Conv2d: 3-12	[128, 256, 28, 28]	16,384
│ │ └─BatchNorm2d: 3-13	[128, 256, 28, 28]	512
│ │ └─Sequential: 3-14	[128, 256, 28, 28]	--
│ └─Bottleneck: 2-3	[128, 256, 28, 28]	--
│ │ └─Conv2d: 3-15	[128, 64, 28, 28]	16,384
│ │ └─BatchNorm2d: 3-16	[128, 64, 28, 28]	128
│ │ └─Conv2d: 3-17	[128, 64, 28, 28]	36,864
│ │ └─BatchNorm2d: 3-18	[128, 64, 28, 28]	128
│ │ └─Conv2d: 3-19	[128, 256, 28, 28]	16,384
│ │ └─BatchNorm2d: 3-20	[128, 256, 28, 28]	512
│ │ └─Sequential: 3-21	[128, 256, 28, 28]	--
├─Sequential: 1-4	[128, 512, 14, 14]	--
│ └─Bottleneck: 2-4	[128, 512, 14, 14]	--
│ │ └─Conv2d: 3-22	[128, 128, 28, 28]	32,768
│ │ └─BatchNorm2d: 3-23	[128, 128, 28, 28]	256
│ │ └─Conv2d: 3-24	[128, 128, 14, 14]	147,456
│ │ └─BatchNorm2d: 3-25	[128, 128, 14, 14]	256
│ │ └─Conv2d: 3-26	[128, 512, 14, 14]	65,536
│ │ └─BatchNorm2d: 3-27	[128, 512, 14, 14]	1,024
│ │ └─Sequential: 3-28	[128, 512, 14, 14]	132,096
│ └─Bottleneck: 2-5	[128, 512, 14, 14]	--
│ │ └─Conv2d: 3-29	[128, 128, 14, 14]	65,536
│ │ └─BatchNorm2d: 3-30	[128, 128, 14, 14]	256
│ │ └─Conv2d: 3-31	[128, 128, 14, 14]	147,456
│ │ └─BatchNorm2d: 3-32	[128, 128, 14, 14]	256
│ │ └─Conv2d: 3-33	[128, 512, 14, 14]	65,536
│ │ └─BatchNorm2d: 3-34	[128, 512, 14, 14]	1,024
│ │ └─Sequential: 3-35	[128, 512, 14, 14]	--
│ └─Bottleneck: 2-6	[128, 512, 14, 14]	--
│ │ └─Conv2d: 3-36	[128, 128, 14, 14]	65,536
│ │ └─BatchNorm2d: 3-37	[128, 128, 14, 14]	256
│ │ └─Conv2d: 3-38	[128, 128, 14, 14]	147,456
│ │ └─BatchNorm2d: 3-39	[128, 128, 14, 14]	256
│ │ └─Conv2d: 3-40	[128, 512, 14, 14]	65,536
│ │ └─BatchNorm2d: 3-41	[128, 512, 14, 14]	1,024
│ │ └─Sequential: 3-42	[128, 512, 14, 14]	--
│ └─Bottleneck: 2-7	[128, 512, 14, 14]	--
│ │ └─Conv2d: 3-43	[128, 128, 14, 14]	65,536
│ │ └─BatchNorm2d: 3-44	[128, 128, 14, 14]	256
│ │ └─Conv2d: 3-45	[128, 128, 14, 14]	147,456
│ │ └─BatchNorm2d: 3-46	[128, 128, 14, 14]	256
│ │ └─Conv2d: 3-47	[128, 512, 14, 14]	65,536
│ │ └─BatchNorm2d: 3-48	[128, 512, 14, 14]	1,024
│ │ └─Sequential: 3-49	[128, 512, 14, 14]	--
├─Sequential: 1-5	[128, 1024, 7, 7]	--
│ └─Bottleneck: 2-8	[128, 1024, 7, 7]	--
│ │ └─Conv2d: 3-50	[128, 256, 14, 14]	131,072
│ │ └─BatchNorm2d: 3-51	[128, 256, 14, 14]	512
│ │ └─Conv2d: 3-52	[128, 256, 7, 7]	589,824
│ │ └─BatchNorm2d: 3-53	[128, 256, 7, 7]	512
│ │ └─Conv2d: 3-54	[128, 1024, 7, 7]	262,144
│ │ └─BatchNorm2d: 3-55	[128, 1024, 7, 7]	2,048
│ │ └─Sequential: 3-56	[128, 1024, 7, 7]	526,336
│ └─Bottleneck: 2-9	[128, 1024, 7, 7]	--
│ │ └─Conv2d: 3-57	[128, 256, 7, 7]	262,144
│ │ └─BatchNorm2d: 3-58	[128, 256, 7, 7]	512
│ │ └─Conv2d: 3-59	[128, 256, 7, 7]	589,824
│ │ └─BatchNorm2d: 3-60	[128, 256, 7, 7]	512
│ │ └─Conv2d: 3-61	[128, 1024, 7, 7]	262,144
│ │ └─BatchNorm2d: 3-62	[128, 1024, 7, 7]	2,048
│ │ └─Sequential: 3-63	[128, 1024, 7, 7]	--
│ └─Bottleneck: 2-10	[128, 1024, 7, 7]	--
│ │ └─Conv2d: 3-64	[128, 256, 7, 7]	262,144
│ │ └─BatchNorm2d: 3-65	[128, 256, 7, 7]	512
│ │ └─Conv2d: 3-66	[128, 256, 7, 7]	589,824
│ │ └─BatchNorm2d: 3-67	[128, 256, 7, 7]	512
│ │ └─Conv2d: 3-68	[128, 1024, 7, 7]	262,144
│ │ └─BatchNorm2d: 3-69	[128, 1024, 7, 7]	2,048
│ │ └─Sequential: 3-70	[128, 1024, 7, 7]	--
│ └─Bottleneck: 2-11	[128, 1024, 7, 7]	--
│ │ └─Conv2d: 3-71	[128, 256, 7, 7]	262,144
│ │ └─BatchNorm2d: 3-72	[128, 256, 7, 7]	512
│ │ └─Conv2d: 3-73	[128, 256, 7, 7]	589,824
│ │ └─BatchNorm2d: 3-74	[128, 256, 7, 7]	512
│ │ └─Conv2d: 3-75	[128, 1024, 7, 7]	262,144
│ │ └─BatchNorm2d: 3-76	[128, 1024, 7, 7]	2,048
│ │ └─Sequential: 3-77	[128, 1024, 7, 7]	--
│ └─Bottleneck: 2-12	[128, 1024, 7, 7]	--
│ │ └─Conv2d: 3-78	[128, 256, 7, 7]	262,144
│ │ └─BatchNorm2d: 3-79	[128, 256, 7, 7]	512
│ │ └─Conv2d: 3-80	[128, 256, 7, 7]	589,824
│ │ └─BatchNorm2d: 3-81	[128, 256, 7, 7]	512
│ │ └─Conv2d: 3-82	[128, 1024, 7, 7]	262,144
│ │ └─BatchNorm2d: 3-83	[128, 1024, 7, 7]	2,048
│ │ └─Sequential: 3-84	[128, 1024, 7, 7]	--
│ └─Bottleneck: 2-13	[128, 1024, 7, 7]	--
│ │ └─Conv2d: 3-85	[128, 256, 7, 7]	262,144
│ │ └─BatchNorm2d: 3-86	[128, 256, 7, 7]	512
│ │ └─Conv2d: 3-87	[128, 256, 7, 7]	589,824
│ │ └─BatchNorm2d: 3-88	[128, 256, 7, 7]	512
│ │ └─Conv2d: 3-89	[128, 1024, 7, 7]	262,144
│ │ └─BatchNorm2d: 3-90	[128, 1024, 7, 7]	2,048
│ │ └─Sequential: 3-91	[128, 1024, 7, 7]	--
├─Sequential: 1-6	[128, 2048, 4, 4]	--
│ └─Bottleneck: 2-14	[128, 2048, 4, 4]	--
│ │ └─Conv2d: 3-92	[128, 512, 7, 7]	524,288
│ │ └─BatchNorm2d: 3-93	[128, 512, 7, 7]	1,024
│ │ └─Conv2d: 3-94	[128, 512, 4, 4]	2,359,296
│ │ └─BatchNorm2d: 3-95	[128, 512, 4, 4]	1,024
│ │ └─Conv2d: 3-96	[128, 2048, 4, 4]	1,048,576
│ │ └─BatchNorm2d: 3-97	[128, 2048, 4, 4]	4,096
│ │ └─Sequential: 3-98	[128, 2048, 4, 4]	2,101,248
│ └─Bottleneck: 2-15	[128, 2048, 4, 4]	--
│ │ └─Conv2d: 3-99	[128, 512, 4, 4]	1,048,576
│ │ └─BatchNorm2d: 3-100	[128, 512, 4, 4]	1,024
│ │ └─Conv2d: 3-101	[128, 512, 4, 4]	2,359,296
│ │ └─BatchNorm2d: 3-102	[128, 512, 4, 4]	1,024
│ │ └─Conv2d: 3-103	[128, 2048, 4, 4]	1,048,576
│ │ └─BatchNorm2d: 3-104	[128, 2048, 4, 4]	4,096
│ │ └─Sequential: 3-105	[128, 2048, 4, 4]	--
│ └─Bottleneck: 2-16	[128, 2048, 4, 4]	--
│ │ └─Conv2d: 3-106	[128, 512, 4, 4]	1,048,576
│ │ └─BatchNorm2d: 3-107	[128, 512, 4, 4]	1,024
│ │ └─Conv2d: 3-108	[128, 512, 4, 4]	2,359,296
│ │ └─BatchNorm2d: 3-109	[128, 512, 4, 4]	1,024
│ │ └─Conv2d: 3-110	[128, 2048, 4, 4]	1,048,576
│ │ └─BatchNorm2d: 3-111	[128, 2048, 4, 4]	4,096
│ │ └─Sequential: 3-112	[128, 2048, 4, 4]	--
├─Linear: 1-7	[128, 9]	18,441

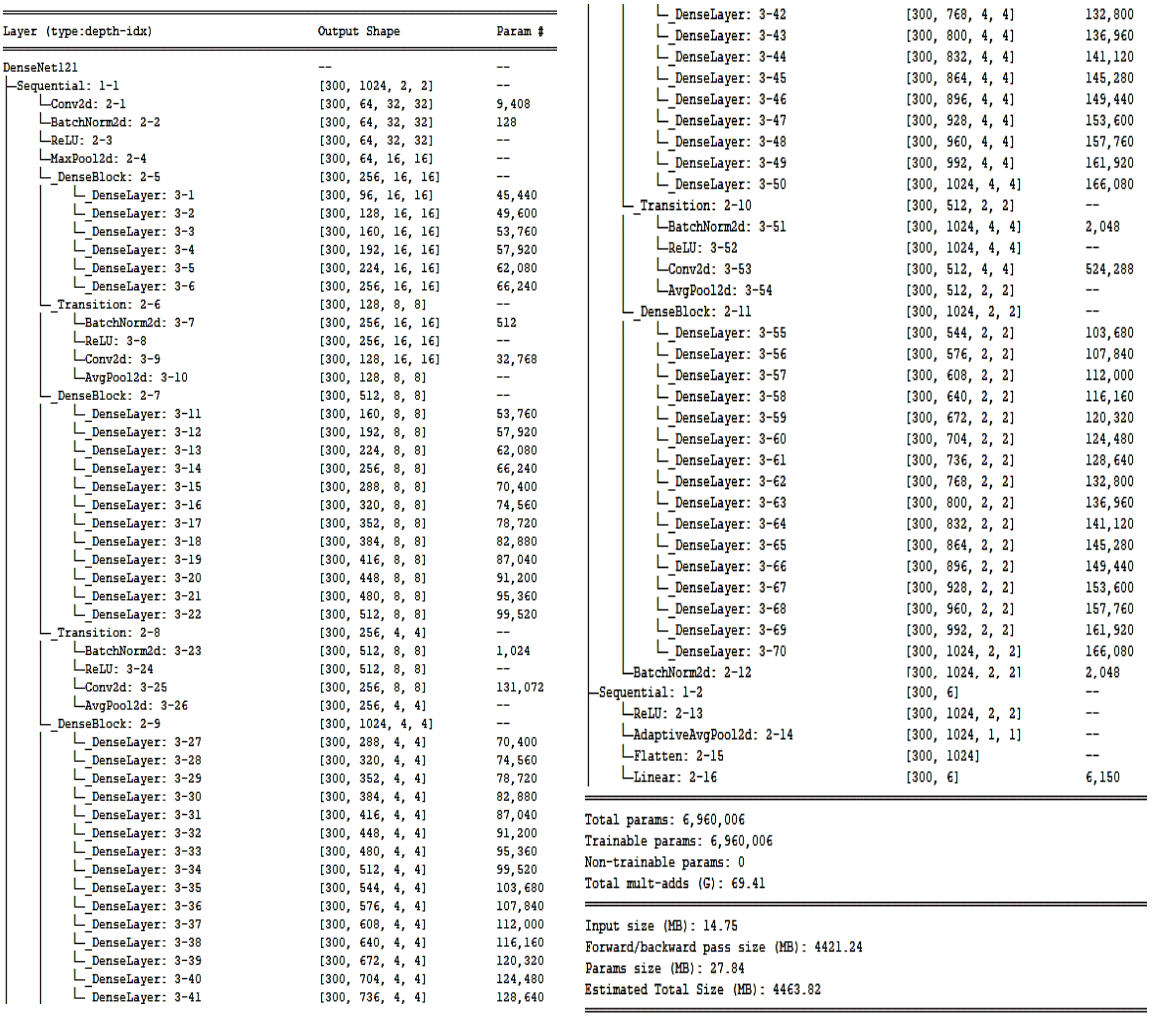
Total params: 23,518,793
Trainable params: 23,518,793
Non-trainable params: 0
Total mult-adds (G): 134.12

Input size (MB): 1.20
Forward/backward pass size (MB): 5464.66
Params size (MB): 94.08
Estimated Total Size (MB): 5559.94

Figura 31. Arquitetura *ResNet50* utilizada no *MedMNIST*.

3.4.4 DenseNet121

A *DenseNet121* implementada para o *MedNIST* (Figura 32) foi construída com os blocos densos, *DenseBlock*, característicos dos modelos *DenseNet* tendo em consideração a arquitetura descrita por Huang *et al.* (2017) [175]. Esta, possui ainda os blocos de transição, *Transition*, para ligar os mapas de características entre a saída de um bloco denso e a entrada do próximo, com a responsabilidade de dimensionar estes mesmos mapas de forma a facilitar o agrupamento de toda a rede. A rede inicia com uma camada de convolução (*Conv2d*), seguida de uma camada de normalização de lotes, uma camada de ativação (função ReLU) e de uma camada de *pooling* máximo. No final da rede foi implementada uma camada de ativação (ReLU), seguida de uma camada de *pooling* médio adaptativo, uma camada de redimensionamento *Flatten* e, por fim, uma camada final totalmente ligada (FC).



Layer (type:depth-idx)	Output Shape	Param #
DenseNet121	--	--
├─Sequential: 1-1	[300, 1024, 2, 2]	--
│ └─Conv2d: 2-1	[300, 64, 32, 32]	9,408
│ └─BatchNorm2d: 2-2	[300, 64, 32, 32]	128
│ └─ReLU: 2-3	[300, 64, 32, 32]	--
│ └─MaxPool2d: 2-4	[300, 64, 16, 16]	--
│ └─_DenseBlock: 2-5	[300, 256, 16, 16]	--
│ │ └─_DenseLayer: 3-1	[300, 96, 16, 16]	45,440
│ │ └─_DenseLayer: 3-2	[300, 128, 16, 16]	49,600
│ │ └─_DenseLayer: 3-3	[300, 160, 16, 16]	53,760
│ │ └─_DenseLayer: 3-4	[300, 192, 16, 16]	57,920
│ │ └─_DenseLayer: 3-5	[300, 224, 16, 16]	62,080
│ │ └─_DenseLayer: 3-6	[300, 256, 16, 16]	66,240
│ └─_Transition: 2-6	[300, 128, 8, 8]	--
│ │ └─BatchNorm2d: 3-7	[300, 256, 16, 16]	512
│ │ └─ReLU: 3-8	[300, 256, 16, 16]	--
│ │ └─Conv2d: 3-9	[300, 128, 16, 16]	32,768
│ │ └─AvgPool2d: 3-10	[300, 128, 8, 8]	--
│ └─_DenseBlock: 2-7	[300, 512, 8, 8]	--
│ │ └─_DenseLayer: 3-11	[300, 160, 8, 8]	53,760
│ │ └─_DenseLayer: 3-12	[300, 192, 8, 8]	57,920
│ │ └─_DenseLayer: 3-13	[300, 224, 8, 8]	62,080
│ │ └─_DenseLayer: 3-14	[300, 256, 8, 8]	66,240
│ │ └─_DenseLayer: 3-15	[300, 288, 8, 8]	70,400
│ │ └─_DenseLayer: 3-16	[300, 320, 8, 8]	74,560
│ │ └─_DenseLayer: 3-17	[300, 352, 8, 8]	78,720
│ │ └─_DenseLayer: 3-18	[300, 384, 8, 8]	82,880
│ │ └─_DenseLayer: 3-19	[300, 416, 8, 8]	87,040
│ │ └─_DenseLayer: 3-20	[300, 448, 8, 8]	91,200
│ │ └─_DenseLayer: 3-21	[300, 480, 8, 8]	95,360
│ │ └─_DenseLayer: 3-22	[300, 512, 8, 8]	99,520
│ └─_Transition: 2-8	[300, 256, 4, 4]	--
│ │ └─BatchNorm2d: 3-23	[300, 512, 8, 8]	1,024
│ │ └─ReLU: 3-24	[300, 512, 8, 8]	--
│ │ └─Conv2d: 3-25	[300, 256, 8, 8]	131,072
│ │ └─AvgPool2d: 3-26	[300, 256, 4, 4]	--
│ └─_DenseBlock: 2-9	[300, 1024, 4, 4]	--
│ │ └─_DenseLayer: 3-27	[300, 288, 4, 4]	70,400
│ │ └─_DenseLayer: 3-28	[300, 320, 4, 4]	74,560
│ │ └─_DenseLayer: 3-29	[300, 352, 4, 4]	78,720
│ │ └─_DenseLayer: 3-30	[300, 384, 4, 4]	82,880
│ │ └─_DenseLayer: 3-31	[300, 416, 4, 4]	87,040
│ │ └─_DenseLayer: 3-32	[300, 448, 4, 4]	91,200
│ │ └─_DenseLayer: 3-33	[300, 480, 4, 4]	95,360
│ │ └─_DenseLayer: 3-34	[300, 512, 4, 4]	99,520
│ │ └─_DenseLayer: 3-35	[300, 544, 4, 4]	103,680
│ │ └─_DenseLayer: 3-36	[300, 576, 4, 4]	107,840
│ │ └─_DenseLayer: 3-37	[300, 608, 4, 4]	112,000
│ │ └─_DenseLayer: 3-38	[300, 640, 4, 4]	116,160
│ │ └─_DenseLayer: 3-39	[300, 672, 4, 4]	120,320
│ │ └─_DenseLayer: 3-40	[300, 704, 4, 4]	124,480
│ │ └─_DenseLayer: 3-41	[300, 736, 4, 4]	128,640
│ │ └─_DenseLayer: 3-42	[300, 768, 4, 4]	132,800
│ │ └─_DenseLayer: 3-43	[300, 800, 4, 4]	136,960
│ │ └─_DenseLayer: 3-44	[300, 832, 4, 4]	141,120
│ │ └─_DenseLayer: 3-45	[300, 864, 4, 4]	145,280
│ │ └─_DenseLayer: 3-46	[300, 896, 4, 4]	149,440
│ │ └─_DenseLayer: 3-47	[300, 928, 4, 4]	153,600
│ │ └─_DenseLayer: 3-48	[300, 960, 4, 4]	157,760
│ │ └─_DenseLayer: 3-49	[300, 992, 4, 4]	161,920
│ │ └─_DenseLayer: 3-50	[300, 1024, 4, 4]	166,080
│ └─_Transition: 2-10	[300, 512, 2, 2]	--
│ │ └─BatchNorm2d: 3-51	[300, 1024, 4, 4]	2,048
│ │ └─ReLU: 3-52	[300, 1024, 4, 4]	--
│ │ └─Conv2d: 3-53	[300, 512, 4, 4]	524,288
│ │ └─AvgPool2d: 3-54	[300, 512, 2, 2]	--
│ └─_DenseBlock: 2-11	[300, 1024, 2, 2]	--
│ │ └─_DenseLayer: 3-55	[300, 544, 2, 2]	103,680
│ │ └─_DenseLayer: 3-56	[300, 576, 2, 2]	107,840
│ │ └─_DenseLayer: 3-57	[300, 608, 2, 2]	112,000
│ │ └─_DenseLayer: 3-58	[300, 640, 2, 2]	116,160
│ │ └─_DenseLayer: 3-59	[300, 672, 2, 2]	120,320
│ │ └─_DenseLayer: 3-60	[300, 704, 2, 2]	124,480
│ │ └─_DenseLayer: 3-61	[300, 736, 2, 2]	128,640
│ │ └─_DenseLayer: 3-62	[300, 768, 2, 2]	132,800
│ │ └─_DenseLayer: 3-63	[300, 800, 2, 2]	136,960
│ │ └─_DenseLayer: 3-64	[300, 832, 2, 2]	141,120
│ │ └─_DenseLayer: 3-65	[300, 864, 2, 2]	145,280
│ │ └─_DenseLayer: 3-66	[300, 896, 2, 2]	149,440
│ │ └─_DenseLayer: 3-67	[300, 928, 2, 2]	153,600
│ │ └─_DenseLayer: 3-68	[300, 960, 2, 2]	157,760
│ │ └─_DenseLayer: 3-69	[300, 992, 2, 2]	161,920
│ │ └─_DenseLayer: 3-70	[300, 1024, 2, 2]	166,080
│ └─BatchNorm2d: 2-12	[300, 1024, 2, 2]	2,048
├─Sequential: 1-2	[300, 6]	--
│ └─ReLU: 2-13	[300, 1024, 2, 2]	--
│ └─AdaptiveAvgPool2d: 2-14	[300, 1024, 1, 1]	--
│ └─Flatten: 2-15	[300, 1024]	--
│ └─Linear: 2-16	[300, 6]	6,150

Total params: 6,960,006
Trainable params: 6,960,006
Non-trainable params: 0
Total mult-adds (G): 69.41

Input size (MB): 14.75
Forward/backward pass size (MB): 4421.24
Params size (MB): 27.84
Estimated Total Size (MB): 4463.82

Figura 32. Arquitetura *DenseNet121* utilizada no *MedNIST.*

Cada um dos quatro blocos densos implementados é composto por camadas densas (Figura 33), *DenseLayer*, onde cada uma destas é composta pela seguinte configuração de camadas: camada de normalização, camada de ativação (ReLU), camada de convolução, camada de normalização, camada de ativação (ReLU), camada de convolução e, ainda, uma camada de *Dropout* para regularizar e prevenir o continua readaptação dos neurónios [138]. O número de camadas densas implementadas em cada bloco denso respeita a seguinte configuração: [6, 12, 24, 16]. A função das perdas implementada consistiu na entropia cruzada (*CrossEntropyLoss*) e a função de otimização foi a *Adam* [226, 227]. A exatidão (ACC) foi a métrica utilizada como medida de desempenho nesta implementação.

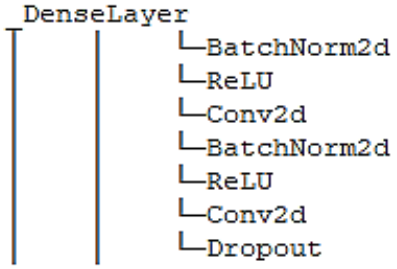
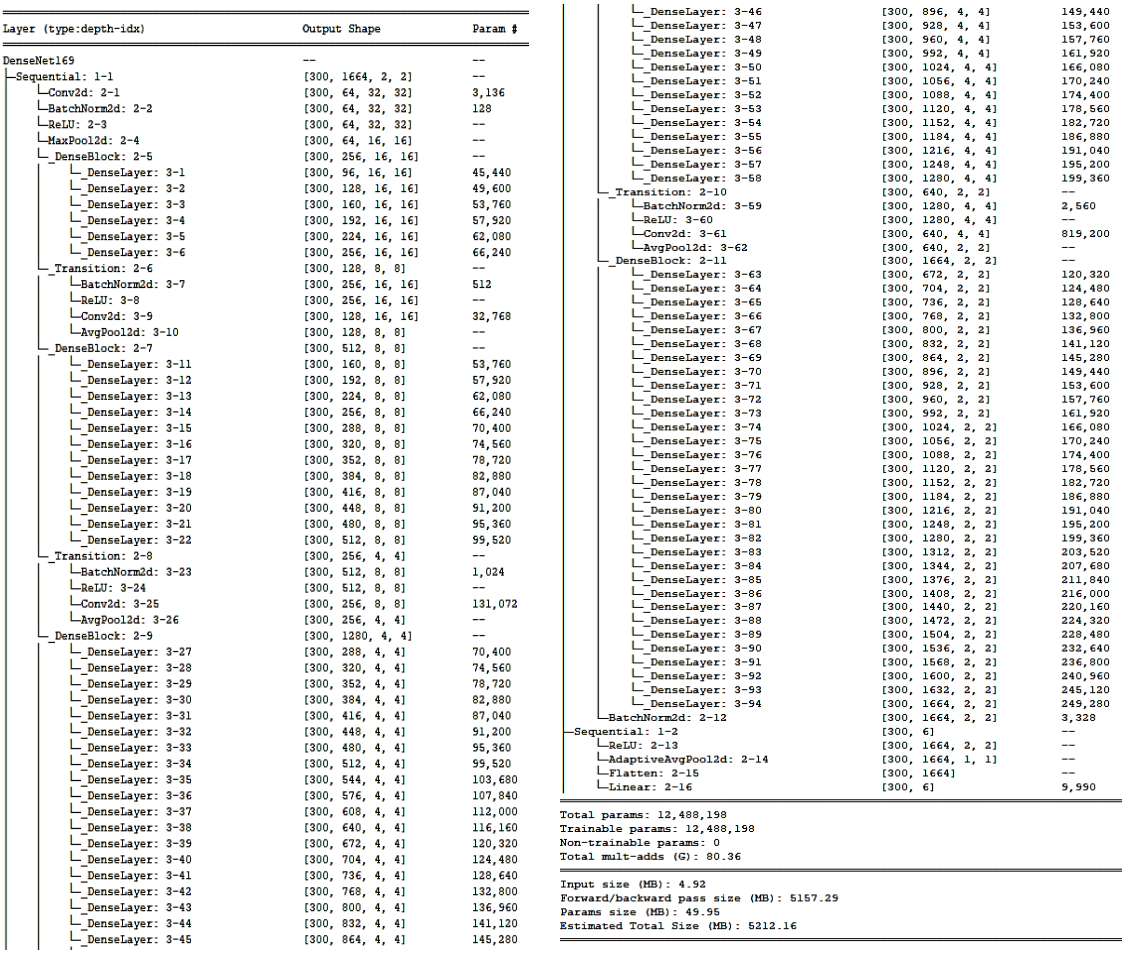


Figura 33. *DenseLayer* utilizada nos blocos densos das *DenseNet.*

3.4.5 DenseNet169

De forma a comparar o desempenho das diferentes variantes CNN no dataset *MedNIST*, foi também implementada a variante *DenseNet169* (Figura 34). Similarmente à *DenseNet121*, a *DenseNet169* implementa os blocos densos (*DenseBlock*), os blocos de transição (*Transition*) e a camadas densas (*DenseLayer*, Figura 33) características das *DenseNet.* O número de camadas densas em cada um dos blocos densos é a principal diferença na arquitetura entre a *DenseNet121* e a *DenseNet169*. Aqui, foi implementada a configuração [6, 12, 32, 32] de camadas densas em cada bloco denso. A métrica de desempenho, função de perdas e otimização usadas nesta implementação foram as mesmas que foram usadas na *DenseNet121*.



Layer (type:depth-idx)	Output Shape	Param #
DenseNet169	--	--
├─Sequential: 1-1	[300, 1664, 2, 2]	--
│ └─Conv2d: 2-1	[300, 64, 32, 32]	3,136
│ └─BatchNorm2d: 2-2	[300, 64, 32, 32]	128
│ └─ReLU: 2-3	[300, 64, 32, 32]	--
│ └─MaxPool2d: 2-4	[300, 64, 16, 16]	--
│ └─_DenseBlock: 2-5	[300, 256, 16, 16]	--
│ │ └─_DenseLayer: 3-1	[300, 96, 16, 16]	45,440
│ │ └─_DenseLayer: 3-2	[300, 128, 16, 16]	49,600
│ │ └─_DenseLayer: 3-3	[300, 160, 16, 16]	53,760
│ │ └─_DenseLayer: 3-4	[300, 192, 16, 16]	57,920
│ │ └─_DenseLayer: 3-5	[300, 224, 16, 16]	62,080
│ │ └─_DenseLayer: 3-6	[300, 256, 16, 16]	66,240
│ └─_Transition: 2-6	[300, 128, 8, 8]	--
│ │ └─BatchNorm2d: 3-7	[300, 256, 16, 16]	512
│ │ └─ReLU: 3-8	[300, 256, 16, 16]	--
│ │ └─Conv2d: 3-9	[300, 128, 16, 16]	32,768
│ │ └─AvgPool2d: 3-10	[300, 128, 8, 8]	--
│ └─_DenseBlock: 2-7	[300, 512, 8, 8]	--
│ │ └─_DenseLayer: 3-11	[300, 160, 8, 8]	53,760
│ │ └─_DenseLayer: 3-12	[300, 192, 8, 8]	57,920
│ │ └─_DenseLayer: 3-13	[300, 224, 8, 8]	62,080
│ │ └─_DenseLayer: 3-14	[300, 256, 8, 8]	66,240
│ │ └─_DenseLayer: 3-15	[300, 288, 8, 8]	70,400
│ │ └─_DenseLayer: 3-16	[300, 320, 8, 8]	74,560
│ │ └─_DenseLayer: 3-17	[300, 352, 8, 8]	78,720
│ │ └─_DenseLayer: 3-18	[300, 384, 8, 8]	82,880
│ │ └─_DenseLayer: 3-19	[300, 416, 8, 8]	87,040
│ │ └─_DenseLayer: 3-20	[300, 448, 8, 8]	91,200
│ │ └─_DenseLayer: 3-21	[300, 480, 8, 8]	95,360
│ │ └─_DenseLayer: 3-22	[300, 512, 8, 8]	99,520
│ └─_Transition: 2-8	[300, 256, 4, 4]	--
│ │ └─BatchNorm2d: 3-23	[300, 512, 8, 8]	1,024
│ │ └─ReLU: 3-24	[300, 512, 8, 8]	--
│ │ └─Conv2d: 3-25	[300, 256, 8, 8]	131,072
│ │ └─AvgPool2d: 3-26	[300, 256, 4, 4]	--
│ └─_DenseBlock: 2-9	[300, 1280, 4, 4]	--
│ │ └─_DenseLayer: 3-27	[300, 288, 4, 4]	70,400
│ │ └─_DenseLayer: 3-28	[300, 320, 4, 4]	74,560
│ │ └─_DenseLayer: 3-29	[300, 352, 4, 4]	78,720
│ │ └─_DenseLayer: 3-30	[300, 384, 4, 4]	82,880
│ │ └─_DenseLayer: 3-31	[300, 416, 4, 4]	87,040
│ │ └─_DenseLayer: 3-32	[300, 448, 4, 4]	91,200
│ │ └─_DenseLayer: 3-33	[300, 480, 4, 4]	95,360
│ │ └─_DenseLayer: 3-34	[300, 512, 4, 4]	99,520
│ │ └─_DenseLayer: 3-35	[300, 544, 4, 4]	103,680
│ │ └─_DenseLayer: 3-36	[300, 576, 4, 4]	107,840
│ │ └─_DenseLayer: 3-37	[300, 608, 4, 4]	112,000
│ │ └─_DenseLayer: 3-38	[300, 640, 4, 4]	116,160
│ │ └─_DenseLayer: 3-39	[300, 672, 4, 4]	120,320
│ │ └─_DenseLayer: 3-40	[300, 704, 4, 4]	124,480
│ │ └─_DenseLayer: 3-41	[300, 736, 4, 4]	128,640
│ │ └─_DenseLayer: 3-42	[300, 768, 4, 4]	132,800
│ │ └─_DenseLayer: 3-43	[300, 800, 4, 4]	136,960
│ │ └─_DenseLayer: 3-44	[300, 832, 4, 4]	141,120
│ │ └─_DenseLayer: 3-45	[300, 864, 4, 4]	145,280
│ │ └─_DenseLayer: 3-46	[300, 896, 4, 4]	149,440
│ │ └─_DenseLayer: 3-47	[300, 928, 4, 4]	153,600
│ │ └─_DenseLayer: 3-48	[300, 960, 4, 4]	157,760
│ │ └─_DenseLayer: 3-49	[300, 992, 4, 4]	161,920
│ │ └─_DenseLayer: 3-50	[300, 1024, 4, 4]	166,080
│ │ └─_DenseLayer: 3-51	[300, 1056, 4, 4]	170,240
│ │ └─_DenseLayer: 3-52	[300, 1088, 4, 4]	174,400
│ │ └─_DenseLayer: 3-53	[300, 1120, 4, 4]	178,560
│ │ └─_DenseLayer: 3-54	[300, 1152, 4, 4]	182,720
│ │ └─_DenseLayer: 3-55	[300, 1184, 4, 4]	186,880
│ │ └─_DenseLayer: 3-56	[300, 1216, 4, 4]	191,040
│ │ └─_DenseLayer: 3-57	[300, 1248, 4, 4]	195,200
│ │ └─_DenseLayer: 3-58	[300, 1280, 4, 4]	199,360
│ └─_Transition: 2-10	[300, 640, 2, 2]	--
│ │ └─BatchNorm2d: 3-59	[300, 1280, 4, 4]	2,560
│ │ └─ReLU: 3-60	[300, 1280, 4, 4]	--
│ │ └─Conv2d: 3-61	[300, 640, 4, 4]	819,200
│ │ └─AvgPool2d: 3-62	[300, 640, 2, 2]	--
│ └─_DenseBlock: 2-11	[300, 1664, 2, 2]	--
│ │ └─_DenseLayer: 3-63	[300, 672, 2, 2]	120,320
│ │ └─_DenseLayer: 3-64	[300, 704, 2, 2]	124,480
│ │ └─_DenseLayer: 3-65	[300, 736, 2, 2]	128,640
│ │ └─_DenseLayer: 3-66	[300, 768, 2, 2]	132,800
│ │ └─_DenseLayer: 3-67	[300, 800, 2, 2]	136,960
│ │ └─_DenseLayer: 3-68	[300, 832, 2, 2]	141,120
│ │ └─_DenseLayer: 3-69	[300, 864, 2, 2]	145,280
│ │ └─_DenseLayer: 3-70	[300, 896, 2, 2]	149,440
│ │ └─_DenseLayer: 3-71	[300, 928, 2, 2]	153,600
│ │ └─_DenseLayer: 3-72	[300, 960, 2, 2]	157,760
│ │ └─_DenseLayer: 3-73	[300, 992, 2, 2]	161,920
│ │ └─_DenseLayer: 3-74	[300, 1024, 2, 2]	166,080
│ │ └─_DenseLayer: 3-75	[300, 1056, 2, 2]	170,240
│ │ └─_DenseLayer: 3-76	[300, 1088, 2, 2]	174,400
│ │ └─_DenseLayer: 3-77	[300, 1120, 2, 2]	178,560
│ │ └─_DenseLayer: 3-78	[300, 1152, 2, 2]	182,720
│ │ └─_DenseLayer: 3-79	[300, 1184, 2, 2]	186,880
│ │ └─_DenseLayer: 3-80	[300, 1216, 2, 2]	191,040
│ │ └─_DenseLayer: 3-81	[300, 1248, 2, 2]	195,200
│ │ └─_DenseLayer: 3-82	[300, 1280, 2, 2]	199,360
│ │ └─_DenseLayer: 3-83	[300, 1312, 2, 2]	203,520
│ │ └─_DenseLayer: 3-84	[300, 1344, 2, 2]	207,680
│ │ └─_DenseLayer: 3-85	[300, 1376, 2, 2]	211,840
│ │ └─_DenseLayer: 3-86	[300, 1408, 2, 2]	216,000
│ │ └─_DenseLayer: 3-87	[300, 1440, 2, 2]	220,160
│ │ └─_DenseLayer: 3-88	[300, 1472, 2, 2]	224,320
│ │ └─_DenseLayer: 3-89	[300, 1504, 2, 2]	228,480
│ │ └─_DenseLayer: 3-90	[300, 1536, 2, 2]	232,640
│ │ └─_DenseLayer: 3-91	[300, 1568, 2, 2]	236,800
│ │ └─_DenseLayer: 3-92	[300, 1600, 2, 2]	240,960
│ │ └─_DenseLayer: 3-93	[300, 1632, 2, 2]	245,120
│ │ └─_DenseLayer: 3-94	[300, 1664, 2, 2]	249,280
│ └─BatchNorm2d: 2-12	[300, 1664, 2, 2]	3,328
├─Sequential: 1-2	[300, 6]	--
│ └─ReLU: 2-13	[300, 1664, 2, 2]	--
│ └─AdaptiveAvgPool2d: 2-14	[300, 1664, 1, 1]	--
│ └─Flatten: 2-15	[300, 1664]	--
│ └─Linear: 2-16	[300, 6]	9,990

Total params: 12,488,198
Trainable params: 12,488,198
Non-trainable params: 0
Total mult-adds (G): 80.36

Input size (MB): 4.92
Forward/backward pass size (MB): 5157.29
Params size (MB): 49.95
Estimated Total Size (MB): 5212.16

Figura 34. Arquitetura *DenseNet169* utilizada no *MedNIST.*

3.4.6 EfficientNet-B0

Ao contrário dos modelos desenvolvidos anteriormente apresentados, a *EfficientNet-B0* foi construída com recurso a *Tensorflow* (tf). A *EfficientNet-B0* construída fundamenta-se na base do modelo desenvolvido por Tan *et al.* (2019) [228], fornecida pela própria plataforma *Tensorflow* (*tf.keras.applications.efficientnet*) [229], com a implementação de algumas camadas adicionais.

As camadas adicionais da *EfficientNet-B0* criada para o *MedNIST* (Figura 35) contemplam uma primeira camada de entrada para redimensionar os dados de entrada (*input_2*), seguida do modelo base *efficientnetb0.* Posteriormente, foi adicionada uma camada *Flatten* (*keras.layers.Flatten()*), uma camada densamente ligada (*tf.keras.layers.Dense()*), uma *Dropout* (*keras.layers.Dropout()*) e, por fim, mais duas camadas densamente ligadas. Na camada *dense* e *dense_1* foi utilizada a função de ativação ReLU e na última camada *dense_2* foi aplicada a função de ativação *Softmax*. Na implementação da *EfficientNet* foi utilizada no cálculo das perdas a entropia cruzada, otimização com recurso à função *Adam* e a exatidão (ACC) como medida de desempenho.

Model: "EfficientNet-B0"

Layer (type:depth-idx)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 64, 64, 3)]	0	
efficientnetb0 (Functional)	(None, 2, 2, 1280)	4049571	input_2[0][0]
flatten (Flatten)	(None, 5120)	0	efficientnetb0[0][0]
dense (Dense)	(None, 512)	2621952	flatten[0][0]
dropout (Dropout)	multiple	0	dense[0][0] dense_1[0][0]
dense_1 (Dense)	(None, 128)	65664	dropout[0][0]
dense_2 (Dense)	(None, 6)	774	dropout[1][0]

Total params: 6,737,961
Trainable params: 6,695,938
Non-trainable params: 42,023

Figura 35. Arquitetura *EfficientNet-B0* utilizada no *MedNIST.*

3.4.7 VGG16

A variante *VGG16* da arquitetura *VGG* foi também implementada de forma a analisar comparativamente o desempenho desta no dataset *MedNIST.* Esta foi construída com base no modelo de Simonyan *et al*. (2014) [135], com recurso a *PyTorch*. A arquitetura da *VGG16* implementada pode ser analisada na Figura 36. Esta contempla um primeiro módulo *sequential,* constituído por um conjunto de várias camadas de convolução, ativação (ReLU) e de *pooling* máximo (para diminuir o espaço dimensional), e um segundo módulo constituído pelas camadas FC, ativação, e camadas de operação de *Dropout*. Entre os dois módulos está presente uma camada de ligação do tipo *pooling* médio adaptativo. O algoritmo de otimização usado neste modelo, durante o processo de treino do *MedNIST,* foi o *Adam* [226]. A função para o cálculo das perdas foi a *CrossEntropyLoss* [224]. A métrica implementada para medida de desempenho foi a exatidão (ACC) (subcapítulo 2.4.1).

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
VGG16                                    --                        --
├─Sequential: 1-1                        [300, 512, 2, 2]          --
│    └─Conv2d: 2-1                       [300, 64, 64, 64]         1,792
│    └─ReLU: 2-2                         [300, 64, 64, 64]         --
│    └─Conv2d: 2-3                       [300, 64, 64, 64]         36,928
│    └─ReLU: 2-4                         [300, 64, 64, 64]         --
│    └─MaxPool2d: 2-5                    [300, 64, 32, 32]         --
│    └─Conv2d: 2-6                       [300, 128, 32, 32]        73,856
│    └─ReLU: 2-7                         [300, 128, 32, 32]        --
│    └─Conv2d: 2-8                       [300, 128, 32, 32]        147,584
│    └─ReLU: 2-9                         [300, 128, 32, 32]        --
│    └─MaxPool2d: 2-10                   [300, 128, 16, 16]        --
│    └─Conv2d: 2-11                      [300, 256, 16, 16]        295,168
│    └─ReLU: 2-12                        [300, 256, 16, 16]        --
│    └─Conv2d: 2-13                      [300, 256, 16, 16]        590,080
│    └─ReLU: 2-14                        [300, 256, 16, 16]        --
│    └─Conv2d: 2-15                      [300, 256, 16, 16]        590,080
│    └─ReLU: 2-16                        [300, 256, 16, 16]        --
│    └─MaxPool2d: 2-17                   [300, 256, 8, 8]          --
│    └─Conv2d: 2-18                      [300, 512, 8, 8]          1,180,160
│    └─ReLU: 2-19                        [300, 512, 8, 8]          --
│    └─Conv2d: 2-20                      [300, 512, 8, 8]          2,359,808
│    └─ReLU: 2-21                        [300, 512, 8, 8]          --
│    └─Conv2d: 2-22                      [300, 512, 8, 8]          2,359,808
│    └─ReLU: 2-23                        [300, 512, 8, 8]          --
│    └─MaxPool2d: 2-24                   [300, 512, 4, 4]          --
│    └─Conv2d: 2-25                      [300, 512, 4, 4]          2,359,808
│    └─ReLU: 2-26                        [300, 512, 4, 4]          --
│    └─Conv2d: 2-27                      [300, 512, 4, 4]          2,359,808
│    └─ReLU: 2-28                        [300, 512, 4, 4]          --
│    └─Conv2d: 2-29                      [300, 512, 4, 4]          2,359,808
│    └─ReLU: 2-30                        [300, 512, 4, 4]          --
│    └─MaxPool2d: 2-31                   [300, 512, 2, 2]          --
├─AdaptiveAvgPool2d: 1-2                 [300, 512, 7, 7]          --
├─Sequential: 1-3                        [300, 1000]               --
│    └─Linear: 2-32                      [300, 4096]               102,764,544
│    └─ReLU: 2-33                        [300, 4096]               --
│    └─Dropout: 2-34                     [300, 4096]               --
│    └─Linear: 2-35                      [300, 4096]               16,781,312
│    └─ReLU: 2-36                        [300, 4096]               --
│    └─Dropout: 2-37                     [300, 4096]               --
│    └─Linear: 2-38                      [300, 1000]               4,097,000
==========================================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
Total mult-adds (G): 413.26
==========================================================================================
Input size (MB): 14.75
Forward/backward pass size (MB): 2676.27
Params size (MB): 553.43
Estimated Total Size (MB): 3244.44
==========================================================================================
```

Figura 36. Arquitetura *VGG16* utilizada no *MedNIST.*

3.5 Segmentação de Imagens

Datasets

Um dos dataset utilizados na implementação de metodologias de deteção e segmentação em imagens médicas foi o conjunto de imagens de núcleos oriundo da iniciativa de competição, denominada por *BBBC038* (ou *DSB2018*) [230, 231].

Com o propósito de identificar núcleos celulares, esta permite a investigadores a deteção de cada célula individual em uma determinada amostra. Depois de identificadas, estas podem ser medidas e estudadas quanto ao seu comportamento quando expostas por exemplo a tratamentos, podendo assim ser compreendidos vários processos biológicos. A segmentação de núcleos em imagens microscópicas é frequentemente o primeiro passo na análise quantitativa de dados de imagem em aplicações biológicas e biomédicas.

O conjunto de imagens é fornecido com os dados de treino e teste já devidamente separados. Este é composto por 670 pares de imagem e respetiva máscara/anotação que serão usados no processo de treino dos modelos implementados. Para teste são fornecidas 65 imagens. A competição fornece ainda segundo conjunto de imagens de teste complementar com 3019 imagens que não será considerado neste trabalho. O conjunto de imagens de treino foi dividido em dois grupos de dados, um para o processo de aprendizagem das redes com 536 imagens e um segundo para o processo de validação inerente do processo de treino com 134 imagens. O dataset é composto por diferentes tipos de imagens histológicas como os tecidos roxos, tecidos roxos e cor-de-rosa, tecidos em escala de cinzentos e os tecidos fluorescentes. Quanto ao tamanho original das imagens, estas variam entre os 256x256 e os 1040x1388.

Um segundo dataset de imagens de núcleos foi utilizado para a tarefa de segmentação e deteção de mitoses. Denominado por *ICPR2012,* este dataset fez parte de um concurso cujo objetivo é detetar mitoses em imagens de laminas coradas de H&E (*Hematoxylin* e *Eosin*) do cancro da mama [232]. A contagem de mitoses é um parâmetro importante para o prognostico do cancro da mama, no entanto a deteção destas é desafiante, dado que a mitose é um pequeno artefacto que pode ser confundido com outros objetos presentes em imagens histológicas. O conjunto de dados é composto por um total de 50 imagens com dimensão de 2048x2048 que se refere a uma área de 512x512 μm. O dataset é dividido pelos autores, com 35 imagens usadas para o treino e 15 imagens para avaliação/teste.

Processamento dos Datasets

- ***DSB2018* e *ICPR2012***

A primeira transformação efetuada ao conjunto de dados *DSB2018* e ao *ICPR2012* consistiu em um redimensionamento de todas as imagens para uma dimensão uniforme de 256x256. Os restantes

métodos de processamento aplicadas aos dataset incluem normalização, inversão e rotações aleatórias de imagem.

3.6 Arquiteturas de Segmentação de Imagem

Na segmentação de núcleos celulares em imagens microscópias foram implementadas diferentes arquiteturas de *Deep Learning* de forma a estudar o desempenho destas na tarefa de identificação e segmentação de núcleos. Os modelos implementados dizem respeito a uma *CNN,* seguida de alguns modelos da família da *UNET*, nomeadamente a *W-UNET* e a *UNET++.*

3.6.1 CNN

A arquitetura CNN desenvolvida para a tarefa de segmentação de núcleos no conjunto de dados *DSB2018* foi construída com recurso aos módulos *sequential* do *Tensorflow* (Figura 37). Esta inicia com uma primeira camada de normalização de lotes (*BatchNorm2d*), seguindo-se sete camadas de convolução, onde a camada *conv2d_3, conv2d_4* e *conv2d_5* inclui dilatações com diferentes amplitudes de forma a aumentar o campo de visão das amostras. A função de ativação implementada consistiu na *Sigmoid* e a função *Adam* foi utilizada como técnica de otimização. A função *DICE* foi usada no cálculo das perdas durante o processo de treino. A exatidão (ACC), interseção sobre a união (IoU), *Recall,* coeficiente de *DICE* e a precisão foram usadas como medidas de desempenho.

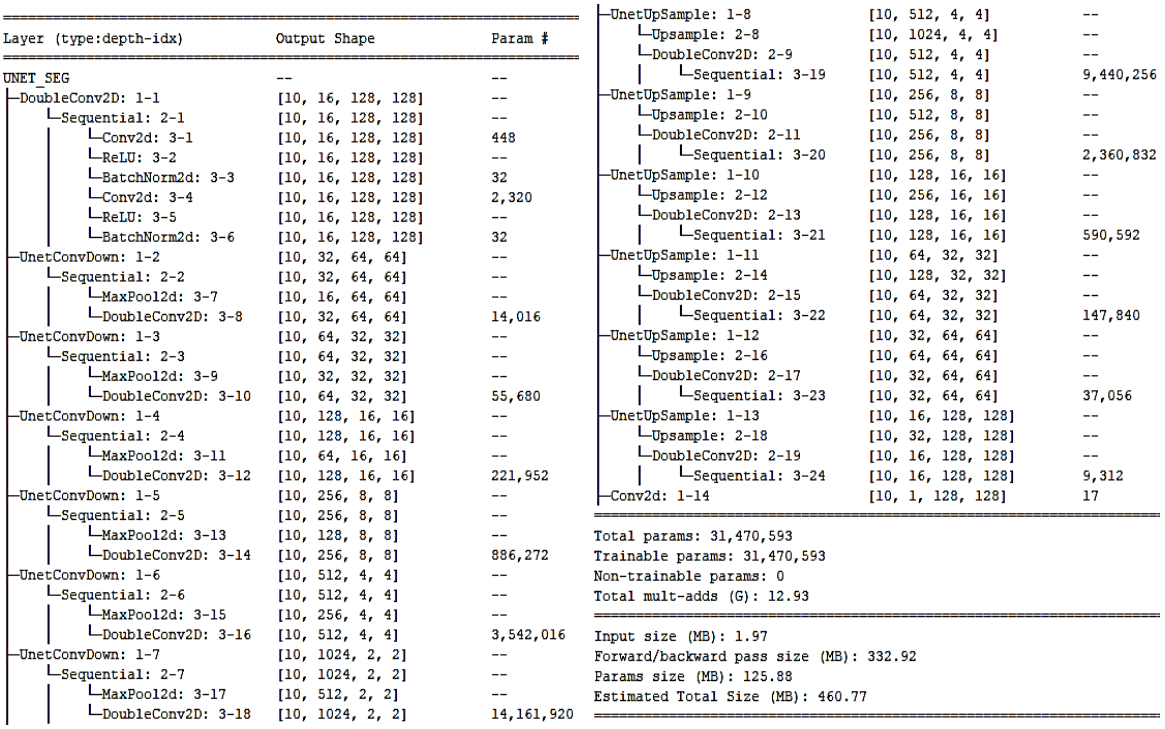
```
Model: CNN_SEG
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
NormalizeInput               (None, 256, 256, 3)       12
└─BatchNorm2d
_________________________________________________________________
conv2d_1                     (None, 256, 256, 8)       224
└─Conv2D
_________________________________________________________________
conv2d_2                     (None, 256, 256, 8)       584
└─Conv2D
_________________________________________________________________
conv2d_3                     (None, 256, 256, 16)      1168
└─Conv2D
_________________________________________________________________
conv2d_4                     (None, 256, 256, 16)      2320
└─Conv2D
_________________________________________________________________
conv2d_5                     (None, 256, 256, 32)      4640
└─Conv2D
_________________________________________________________________
conv2d_6                     (None, 256, 256, 16)      528
└─Conv2D
_________________________________________________________________
conv2d_7                     (None, 256, 256, 1)       17
└─Conv2D
    └─Sigmoid
=================================================================
Total params: 9,493
Trainable params: 9,487
Non-trainable params: 6
_________________________________________________________________
```

Figura 37. Arquitetura CNN utilizada no *DSB2018.*

3.6.2 U-NET

A arquitetura *U-NET* (subcapítulo 2.4.6.7), desenvolvida para a segmentação de núcleos, foi construída com recurso a *PyTorch* através da implementação de camadas agrupadas em diferentes módulos/ blocos (Figura 38). Esta implementação começa com um primeiro bloco *DoubleConv2D*, seguido de seis blocos denominados por *UnetConvDown* (Figura 38). Estes são os blocos responsáveis pelo movimento de *DownSampling* da rede. O bloco *DoubleConv2D* é composto por duas séries de camadas de convolução, de ativação (ReLU) e de normalização: [*Conv2d; ReLU; BatchNorm2d; Conv2d; ReLU; BatchNorm2d*]. Em relação à *UnetConvDown*, esta contempla uma primeira camada de *pooling* máximo, seguido de um bloco *DoubleConv2D.* Para o movimento de *UpSampling* da rede, foram utilizados seis blocos denominados por *UnetUpSample*. Estes, são constituídos por uma camada *Upsample* [233] (com um fator de escala de dois, modo bilinear e com alinhamento de extremos) e por um bloco *DoubleConv2D.* A arquitetura termina com uma camada de convolução, seguida da camada de ativação com implementação da *Sigmoid* (Tabela 1, No. 1). Durante a implementação da *U-NET*foram testadas diferentes funções de otimização, a destacar a *Adam* e a descida de gradiente estocástico. As métricas de desempenho utilizadas foram a exatidão (ACC), coeficiente *DICE*, interseção sobre a união (IoU), *Recall* e a precisão. As funções utilizadas no cálculo das perdas incluem a entropia cruzada binária e o coeficiente de *DICE* [67].

A utilização do *DICE* é pertinente em segmentação dado que em imagiologia médica é frequente as regiões de interesse serem pequenas em comparação com toda a amostra, levando o processo de aprendizagem a paralisar nos mínimos locais da função de perda [234].



Layer (type:depth-idx)	Output Shape	Param #
UNET_SEG	--	--
├─DoubleConv2D: 1-1	[10, 16, 128, 128]	--
│ └─Sequential: 2-1	[10, 16, 128, 128]	--
│ │ └─Conv2d: 3-1	[10, 16, 128, 128]	448
│ │ └─ReLU: 3-2	[10, 16, 128, 128]	--
│ │ └─BatchNorm2d: 3-3	[10, 16, 128, 128]	32
│ │ └─Conv2d: 3-4	[10, 16, 128, 128]	2,320
│ │ └─ReLU: 3-5	[10, 16, 128, 128]	--
│ │ └─BatchNorm2d: 3-6	[10, 16, 128, 128]	32
├─UnetConvDown: 1-2	[10, 32, 64, 64]	--
│ └─Sequential: 2-2	[10, 32, 64, 64]	--
│ │ └─MaxPool2d: 3-7	[10, 16, 64, 64]	--
│ │ └─DoubleConv2D: 3-8	[10, 32, 64, 64]	14,016
├─UnetConvDown: 1-3	[10, 64, 32, 32]	--
│ └─Sequential: 2-3	[10, 64, 32, 32]	--
│ │ └─MaxPool2d: 3-9	[10, 32, 32, 32]	--
│ │ └─DoubleConv2D: 3-10	[10, 64, 32, 32]	55,680
├─UnetConvDown: 1-4	[10, 128, 16, 16]	--
│ └─Sequential: 2-4	[10, 128, 16, 16]	--
│ │ └─MaxPool2d: 3-11	[10, 64, 16, 16]	--
│ │ └─DoubleConv2D: 3-12	[10, 128, 16, 16]	221,952
├─UnetConvDown: 1-5	[10, 256, 8, 8]	--
│ └─Sequential: 2-5	[10, 256, 8, 8]	--
│ │ └─MaxPool2d: 3-13	[10, 128, 8, 8]	--
│ │ └─DoubleConv2D: 3-14	[10, 256, 8, 8]	886,272
├─UnetConvDown: 1-6	[10, 512, 4, 4]	--
│ └─Sequential: 2-6	[10, 512, 4, 4]	--
│ │ └─MaxPool2d: 3-15	[10, 256, 4, 4]	--
│ │ └─DoubleConv2D: 3-16	[10, 512, 4, 4]	3,542,016
├─UnetConvDown: 1-7	[10, 1024, 2, 2]	--
│ └─Sequential: 2-7	[10, 1024, 2, 2]	--
│ │ └─MaxPool2d: 3-17	[10, 512, 2, 2]	--
│ │ └─DoubleConv2D: 3-18	[10, 1024, 2, 2]	14,161,920
├─UnetUpSample: 1-8	[10, 512, 4, 4]	--
│ └─Upsample: 2-8	[10, 1024, 4, 4]	--
│ └─DoubleConv2D: 2-9	[10, 512, 4, 4]	--
│ │ └─Sequential: 3-19	[10, 512, 4, 4]	9,440,256
├─UnetUpSample: 1-9	[10, 256, 8, 8]	--
│ └─Upsample: 2-10	[10, 512, 8, 8]	--
│ └─DoubleConv2D: 2-11	[10, 256, 8, 8]	--
│ │ └─Sequential: 3-20	[10, 256, 8, 8]	2,360,832
├─UnetUpSample: 1-10	[10, 128, 16, 16]	--
│ └─Upsample: 2-12	[10, 256, 16, 16]	--
│ └─DoubleConv2D: 2-13	[10, 128, 16, 16]	--
│ │ └─Sequential: 3-21	[10, 128, 16, 16]	590,592
├─UnetUpSample: 1-11	[10, 64, 32, 32]	--
│ └─Upsample: 2-14	[10, 128, 32, 32]	--
│ └─DoubleConv2D: 2-15	[10, 64, 32, 32]	--
│ │ └─Sequential: 3-22	[10, 64, 32, 32]	147,840
├─UnetUpSample: 1-12	[10, 32, 64, 64]	--
│ └─Upsample: 2-16	[10, 64, 64, 64]	--
│ └─DoubleConv2D: 2-17	[10, 32, 64, 64]	--
│ │ └─Sequential: 3-23	[10, 32, 64, 64]	37,056
├─UnetUpSample: 1-13	[10, 16, 128, 128]	--
│ └─Upsample: 2-18	[10, 32, 128, 128]	--
│ └─DoubleConv2D: 2-19	[10, 16, 128, 128]	--
│ │ └─Sequential: 3-24	[10, 16, 128, 128]	9,312
├─Conv2d: 1-14	[10, 1, 128, 128]	17

Total params: 31,470,593
Trainable params: 31,470,593
Non-trainable params: 0
Total mult-adds (G): 12.93

Input size (MB): 1.97
Forward/backward pass size (MB): 332.92
Params size (MB): 125.88
Estimated Total Size (MB): 460.77

Figura 38. Arquitetura *U-NET* utilizada no *DSB2018*.

3.6.3 W-UNET

De forma a comparar os resultados obtidos em tarefas de segmentação com recurso às arquiteturas da família da *U-NET*, foi implementada a variante denominada por *W-UNET* também conhecida por *W-NET* [235]. A *W-NET* é constituída por duas *U-NET*, onde a primeira funciona como codificador dos valores de entrada, ficando a descodificação a cargo da segunda *U-NET* onde os valores são reconstruídos.

A *W-UNET* implementada é composta por dois tipos de módulos, o bloco *Downsampling* que inclui duas camadas de convolução e uma última camada de *pooling* máximo. O segundo módulo é responsável pela funcionalidade de *Upsampling* e é constituído por uma camada de convolução transposta (operação de deconvolução) concatenada com um módulo *Downsampling* [236]. No total, foram implementados quatro módulos *Downsampling* e quatro módulos de *Upsample*. Entre cada uma das camadas de convolução é usada uma camada de ativação ReLU seguida de uma camada de normalização (Figura 39). À semelhança com as anteriores implementações para a tarefa de segmentação, as perdas foram calculadas com a função *DICE,* e as métricas utilizadas foram a exatidão (ACC), IoU, coeficiente de *DICE*, *Recall* e a precisão.

Model: W_UNET

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)	0	
conv2d (Conv2D)	(None, 256, 256, 35)	980	input_1[0][0]
conv2d_1 (Conv2D)	(None, 256, 256, 35)	11060	conv2d[0][0]
max_pooling2d (MaxPooling2D)	(None, 128, 128, 35)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 70)	22120	max_pooling2d[0][0]
conv2d_3 (Conv2D)	(None, 128, 128, 70)	44170	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 70)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 140)	88340	max_pooling2d_1[0][0]
conv2d_5 (Conv2D)	(None, 64, 64, 140)	176540	conv2d_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 140)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 32, 32, 280)	353080	max_pooling2d_2[0][0]
conv2d_7 (Conv2D)	(None, 32, 32, 280)	705880	conv2d_6[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 280)	0	conv2d_7[0][0]
conv2d_8 (Conv2D)	(None, 16, 16, 560)	1411760	max_pooling2d_3[0][0]
conv2d_9 (Conv2D)	(None, 16, 16, 560)	2822960	conv2d_8[0][0]
conv2d_transpose (Conv2DTranspose)	(None, 32, 32, 280)	627480	conv2d_9[0][0]
concatenate (Concatenate)	(None, 32, 32, 560)	0	conv2d_transpose[0][0] conv2d_7[0][0]
conv2d_10 (Conv2D)	(None, 32, 32, 280)	1411480	concatenate[0][0]
conv2d_11 (Conv2D)	(None, 32, 32, 280)	705880	conv2d_10[0][0]
conv2d_transpose_1 (Conv2DTranspose)	(None, 64, 64, 140)	156940	conv2d_11[0][0]
concatenate_1 (Concatenate)	(None, 64, 64, 280)	0	conv2d_transpose_1[0][0] conv2d_5[0][0]
conv2d_12 (Conv2D)	(None, 64, 64, 140)	352940	concatenate_1[0][0]
conv2d_13 (Conv2D)	(None, 64, 64, 140)	176540	conv2d_12[0][0]
conv2d_transpose_2 (Conv2DTrans	(None, 128, 128, 70)	39270	conv2d_13[0][0]
concatenate_2 (Concatenate)	(None, 128, 128, 140	0	conv2d_transpose_2[0][0] conv2d_3[0][0]
conv2d_14 (Conv2D)	(None, 128, 128, 70)	88270	concatenate_2[0][0]
conv2d_15 (Conv2D)	(None, 128, 128, 70)	44170	conv2d_14[0][0]
conv2d_transpose_3 (Conv2DTranspose)	(None, 256, 256, 35)	9835	conv2d_15[0][0]
concatenate_3 (Concatenate)	(None, 256, 256, 70)	0	conv2d_transpose_3[0][0] conv2d_1[0][0]
conv2d_16 (Conv2D)	(None, 256, 256, 35)	22085	concatenate_3[0][0]
conv2d_17 (Conv2D)	(None, 256, 256, 35)	11060	conv2d_16[0][0]
conv2d_18 (Conv2D)	(None, 256, 256, 1)	36	conv2d_17[0][0]

Total params: 9,282,876
Trainable params: 9,282,876
Non-trainable params: 0

Figura 39. Arquitetura *W_UNET* utilizada no *DSB2018.*

3.6.4 UNET++

A variante *UNET++* foi também implementada de forma a complementar o estudo das *U-NET* na tarefa de segmentação de núcleos em imagens microscópicas. Também conhecida por *Nested-UNET* esta é uma nova arquitetura usada em segmentação de imagens sendo composta por duas redes *U-NET* de diferentes profundidades cujos descodificadores estão densamente ligados à mesma resolução através de vias de fuga [198].

A *UNET++* implementada foi construída com recurso aos módulos *Upsampling* e *Downsampling*, onde o *Downsampling* à semelhança com a *W-UNET* é composto por duas camadas de convolução seguidas de uma camada de *polling* máximo (Figura 40). A rede inicia com quatro módulos de *Downsampling*, seguindo-se um conjunto de 10 módulos de *Upsampling* composto pela convolução transposta, uma concatenação entre a saída da convolução e uma saída das unidades de *Downsampling* e por fim uma unidade de *Downsampling* adicional (para a saída desta ser concatenada na próxima série *Upsampling*). A saída da rede é composta por uma série de quatro camadas de convolução com ativação *sigmoid*. A função de otimização usada foi a *Adam*, as perdas foram calculadas com recurso à função *DICE* e as métricas implementadas foram a exatidão (ACC), coeficiente de DICE, *Recall*, IoU e a precisão.

Model: UNET++

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)	0	
conv2d (Conv2D)	(None, 256, 256, 16)	448	input_1[0][0]
conv2d_1 (Conv2D)	(None, 256, 256, 16)	2320	conv2d[0][0]
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 128, 128, 32)	4640	max_pooling2d[0][0]
conv2d_3 (Conv2D)	(None, 128, 128, 32)	9248	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 32)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 64, 64, 64)	18496	max_pooling2d_1[0][0]
conv2d_5 (Conv2D)	(None, 64, 64, 64)	36928	conv2d_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 32, 32, 128)	73856	max_pooling2d_2[0][0]
conv2d_7 (Conv2D)	(None, 32, 32, 128)	147584	conv2d_6[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0	conv2d_7[0][0]
conv2d_8 (Conv2D)	(None, 16, 16, 256)	295168	max_pooling2d_3[0][0]
conv2d_9 (Conv2D)	(None, 16, 16, 256)	590080	conv2d_8[0][0]
conv2d_transpose_3 (Conv2DTranspose)	(None, 32, 32, 128)	131200	conv2d_9[0][0]
concatenate_3 (Concatenate)	(None, 32, 32, 256)	0	conv2d_transpose_3[0][0] conv2d_7[0][0]
conv2d_transpose_2 (Conv2DTranspose)	(None, 64, 64, 64)	32832	conv2d_7[0][0]
conv2d_16 (Conv2D)	(None, 32, 32, 128)	295040	concatenate_3[0][0]
concatenate_2 (Concatenate)	(None, 64, 64, 128)	0	conv2d_transpose_2[0][0] conv2d_5[0][0]
conv2d_transpose_1 (Conv2DTranspose)	(None, 128, 128, 32)	8224	conv2d_5[0][0]
conv2d_17 (Conv2D)	(None, 32, 32, 128)	147584	conv2d_16[0][0]
conv2d_14 (Conv2D)	(None, 64, 64, 64)	73792	concatenate_2[0][0]
concatenate_1 (Concatenate)	(None, 128, 128, 64)	0	conv2d_transpose_1[0][0] conv2d_3[0][0]
conv2d_transpose (Conv2DTranspose)	(None, 256, 256, 16)	2064	conv2d_3[0][0]
conv2d_transpose_6 (Conv2DTranspose)	(None, 64, 64, 64)	32832	conv2d_17[0][0]
conv2d_15 (Conv2D)	(None, 64, 64, 64)	36928	conv2d_14[0][0]
conv2d_12 (Conv2D)	(None, 128, 128, 32)	18464	concatenate_1[0][0]
concatenate (Concatenate)	(None, 256, 256, 32)	0	conv2d_transpose[0][0] conv2d_1[0][0]
concatenate_6 (Concatenate)	(None, 64, 64, 192)	0	conv2d_transpose_6[0][0] conv2d_5[0][0] conv2d_15[0][0]
conv2d_13 (Conv2D)	(None, 128, 128, 32)	9248	conv2d_12[0][0]
conv2d_transpose_5 (Conv2DTranspose)	(None, 128, 128, 32)	8224	conv2d_15[0][0]
conv2d_10 (Conv2D)	(None, 256, 256, 16)	4624	concatenate[0][0]
conv2d_22 (Conv2D)	(None, 64, 64, 64)	110656	concatenate_6[0][0]
concatenate_5 (Concatenate)	(None, 128, 128, 96)	0	conv2d_transpose_5[0][0] conv2d_3[0][0] conv2d_13[0][0]
conv2d_11 (Conv2D)	(None, 256, 256, 16)	2320	conv2d_10[0][0]
conv2d_transpose_4 (Conv2DTranspose)	(None, 256, 256, 16)	2064	conv2d_13[0][0]
conv2d_23 (Conv2D)	(None, 64, 64, 64)	36928	conv2d_22[0][0]
conv2d_20 (Conv2D)	(None, 128, 128, 32)	27680	concatenate_5[0][0]
concatenate_4 (Concatenate)	(None, 256, 256, 48)	0	conv2d_transpose_4[0][0] conv2d_1[0][0] conv2d_11[0][0]
conv2d_transpose_8 (Conv2DTranspose)	(None, 128, 128, 32)	8224	conv2d_23[0][0]
conv2d_21 (Conv2D)	(None, 128, 128, 32)	9248	conv2d_20[0][0]
conv2d_18 (Conv2D)	(None, 256, 256, 16)	6928	concatenate_4[0][0]
concatenate_8 (Concatenate)	(None, 128, 128, 128	0	conv2d_transpose_8[0][0] conv2d_3[0][0] conv2d_13[0][0] conv2d_21[0][0]
conv2d_19 (Conv2D)	(None, 256, 256, 16)	2320	conv2d_18[0][0]
conv2d_transpose_7 (Conv2DTranspose)	(None, 256, 256, 16)	2064	conv2d_21[0][0]
conv2d_26 (Conv2D)	(None, 128, 128, 32)	36896	concatenate_8[0][0]
concatenate_7 (Concatenate)	(None, 256, 256, 64)	0	conv2d_transpose_7[0][0] conv2d_1[0][0] conv2d_11[0][0] conv2d_19[0][0]
conv2d_27 (Conv2D)	(None, 128, 128, 32)	9248	conv2d_26[0][0]
conv2d_24 (Conv2D)	(None, 256, 256, 16)	9232	concatenate_7[0][0]
conv2d_transpose_9 (Conv2DTranspose)	(None, 256, 256, 16)	2064	conv2d_27[0][0]
conv2d_25 (Conv2D)	(None, 256, 256, 16)	2320	conv2d_24[0][0]
concatenate_9 (Concatenate)	(None, 256, 256, 80)	0	conv2d_transpose_9[0][0] conv2d_1[0][0] conv2d_11[0][0] conv2d_19[0][0] conv2d_25[0][0]
conv2d_28 (Conv2D)	(None, 256, 256, 16)	11536	concatenate_9[0][0]
conv2d_29 (Conv2D)	(None, 256, 256, 16)	2320	conv2d_28[0][0]
conv2d_33 (Conv2D)	(None, 256, 256, 1)	17	conv2d_29[0][0]

Total params: 2,261,889
Trainable params: 2,261,889
Non-trainable params: 0

Figura 40. Arquitetura *UNET++* utilizada no *DSB2018.*

3.7 Aplicação Web: Demonstração de resultados

Como ferramenta de demonstração de resultados foi desenvolvida uma aplicação web, para testar os modelos criados em cada um dos conjuntos de dados utilizados durante este trabalho. Para tal, foi utilizada a biblioteca *streamlit.* Esta aplicação web demonstra e promove um fácil acesso aos desenvolvimentos realizados durante este trabalho indiciando uma rápida transição para uma aplicação em contexto real de aplicação.

Para testar um modelo, por exemplo para visualizar resultados ou para verificar os seus níveis de desempenho, este precisa, inicialmente, de passar pelo processo de treino. Este processo é muitas vezes demorado, o que em uma aplicação web não é o pretendido. De modo a obter os resultados dos modelos, sem que seja necessário passar pelo processo de aprendizagem, os modelos foram guardados durante a execução do processo de treino para que depois possam ser utilizados apenas em operações de teste.

Capítulo 4: Resultados e Discussão

4.1 Classificação de Imagens

Após a aplicação das metodologias de pré-processamento de imagem e implementadas as diferentes arquiteturas, os conjuntos de dados foram submetidos ao processo de treino através das configurações planeadas. Neste subcapítulo serão apresentados e discutidos os resultados obtidos para a classificação com os dataset´s *MedMNIST* e *MedNIST*.

4.1.1 Resultados de classificação obtidos com o Dataset MedMNIST

Para a classificação de imagens médicas, com o conjunto de dados *MedMNIST*, foram criadas diferentes configurações, em adição a quatro configurações previamente reportadas por Yang *et al.* (2020) [237]. Estas configurações variam entre si no modelo utilizado, no número de iterações e nas transformações efetuadas aos dados. Na Tabela 7 estão sumarizadas as informações de cada uma das configurações desenvolvidas (configurações 5-20), assim como, as configurações previamente reportadas na literatura (configurações 1-4) [237], que serviram como controlo comparativo ao longo do estudo. As transformações a que estas foram sujeitas não são mencionadas na literatura. Todas as configurações têm em comum a taxa de aprendizagem (*learning rate*, LR) e o fator de dinâmica (*momentum*) que são parâmetros posteriormente utilizados na função de otimização SGD. A avaliação do desempenho de cada uma das configurações desenvolvidas teve como base a métrica de exatidão (ACC) e a da área sob a curva ROC (AUC). Estas foram selecionadas por serem as mais indicadas para este tipo de tarefa e por serem as reportadas por Yang *et al.* (2020) [237], permitindo assim o estudo comparativo. Pelas mesmas razões foram selecionadas as funções utilizadas no cálculo de perdas.

Tabela 7. Configurações aplicadas no dataset *MedMNIST* para a classificação de imagens.

Configuração	Modelo	*Batch*	*Epochs*	Resolução	Transformações	LR	Fator de dinâmica (*momentum*)	Otimização
1	ResNet18	128	100	28	-	0.001	0.9	SGD [LR, *momentum*]
2	ResNet18	128	100	224				
3	ResNet50	128	100	28				
4	ResNet50	128	100	224				
5	CNN	128	5	28	2			
6	ResNet18	128	5	28	2			
7	ResNet50	128	5	28	2			
8	ResNet18	128	50	28	2			
9	ResNet50	128	50	28	2			
10	ResNet18	256	5	28	2			
11	ResNet18	256	5	28	1			
12	ResNet18	128	5	28	1			
13	ResNet18	128	5	28	3			
14	ResNet18	128	50	28	3			
15	ResNet18	128	100	28	3			
16	ResNet18	128	100	28	2			
17	CNN	128	100	28	2			
18	CNN	128	50	28	4			
19	ResNet18	128	100	28	4			
20	CNN	128	100	28	4			

Transformações	
1	Converter imagem em Tensor
2	Converter imagem em Tensor; Normalização de imagem (média de 0.5 e desvio de 1.0)
3	Converter imagem em Tensor; Normalização de imagem (média de 1.0 e desvio de 1.0)
4	Converter imagem em Tensor; Normalização de imagem (média de 0.5 e desvio de 1.0); Aplicação de rotações aleatórias; Dimensionamento de intensidade da imagem

Nas Tabela 8-Tabela **12** estão presentes os resultados obtidos nas diferentes configurações desenvolvidas, para cada uma das classes presentes no conjunto de dados *MedMNIST*. Estes resultados estão apresentados por ondem crescente de exatidão, *accuracy* (ACC), em um gradiente de cores, que varia entre vermelho (valores mais baixos de ACC) e verde (valores mais altos de ACC) (Figura 41). Os resultados obtidos para as configurações 1-4 correspondem, tal como as configurações, a resultados previamente reportados na literatura [237]. Estes dados foram, também, utilizados como controlo para o trabalho desenvolvido com o dataset *MedMNIST*.

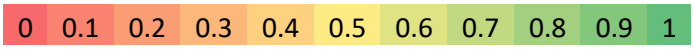


Figura 41. Gradiente de cores para apresentação dos valores das métricas.

Na Tabela 8 estão apresentados os resultados referentes às classes PathMNIST e OCTMNIST. Para a classe PathMNIST, as configurações que obtiveram os melhores resultados foram a 19 e a 20, sendo estes equivalentes aos resultados previamente reportados (configuração 2) [237]. A diferença entre estas consiste apenas no modelo utilizado. Interessantemente, foram estas mesmas configurações (19 e 20) que apresentaram os melhores resultados para a classe OCTMNIST, mas, para esta classe, as configurações criadas apresentaram resultados superiores aos da literatura [237].

Tabela 8. Resultados obtidos no processo de aprendizagem com as arquiteturas desenvolvidas, para as classes PathMNIST e OCTMNIST do dataset MedMNIST.

PathMNIST					**OCTMNIST**				
Configuração	**Modelo**	***Epochs***	**AUC**	**ACC**	**Configuração**	**Modelo**	***Epochs***	**AUC**	**ACC**
5	CNN	5	0.944	0.649	**11**	ResNet18	5	0.939	0.687
11	ResNet18	5	0.966	0.719	**17**	CNN	100	0.935	0.692
10	ResNet18	5	0.966	0.723	**5**	CNN	5	0.943	0.706
12	ResNet18	5	0.970	0.725	**15**	ResNet18	100	0.945	0.721
17	CNN	100	0.967	0.731	**10**	ResNet18	5	0.951	0.723
7	ResNet50	5	0.969	0.737	**12**	ResNet18	5	0.950	0.724
14	ResNet18	50	0.969	0.739	**13**	ResNet18	5	0.950	0.725
8	ResNet18	50	0.969	0.739	**16**	ResNet18	100	0.947	0.742
6	ResNet18	5	0.967	0.741	**3**	ResNet50	100	0.939	0.745
3	ResNet50	100	0.968	0.746	**14**	ResNet18	50	0.951	0.748
16	ResNet18	100	0.969	0.749	**4**	ResNet50	100	0.951	0.750
9	ResNet50	50	0.968	0.752	**2**	ResNet18	100	0.960	0.752
15	ResNet18	100	0.969	0.755	**6**	ResNet18	5	0.957	0.757
13	ResNet18	5	0.969	0.756	**1**	ResNet18	100	0.951	0.758
1	ResNet18	100	0.967	0.762	**8**	ResNet18	50	0.953	0.759
4	ResNet50	100	0.970	0.770	**9**	ResNet50	5	0.952	0.770
18	CNN	50	0.970	0.772	**18**	CNN	50	0.956	0.773
20	CNN	100	0.974	0.775	**7**	ResNet50	50	0.957	0.775
2	ResNet18	100	0.974	0.777	**20**	CNN	100	0.958	0.775
19	ResNet18	100	0.975	0.777	**19**	ResNet18	100	0.957	0.777

AUC = área sobre a curva (*Area under curve*); ACC = Exatidão (*Accuracy*)

Os resultados da classificação obtidos para as classes ChestMNIST e PneumoniaMNIST estão presentes na Tabela 9. Tal como para as classes anteriores, a configuração 19 apresenta os resultados mais relevantes para ambas as classes, sendo que os valores de ACC são equivalentes aos obtidos para uma das configurações previamente reportada (configuração 2), ou consideravelmente melhores no caso da classe PneumoniaMNIST. Contudo, o mesmo não se verifica para a configuração 20, na classe ChestMNIST.

Tabela 9. Resultados obtidos no processo de aprendizagem com as arquiteturas desenvolvidas, para as classes ChestMNIST e PneumoniaMNIST do dataset MedMNIST

ChestMNIST					**PneumoniaMNIST**				
Configuração	**Modelo**	***Epochs***	**AUC**	**ACC**	**Configuração**	**Modelo**	***Epochs***	**AUC**	**ACC**
20	CNN	100	0.703	0.946	**10**	ResNet18	5	0.958	0.803
1	ResNet18	100	0.706	0.947	**13**	ResNet18	5	0.961	0.822
3	ResNet50	100	0.692	0.947	**6**	ResNet18	5	0.958	0.832
4	ResNet50	100	0.706	0.947	**9**	ResNet50	5	0.960	0.833
18	CNN	50	0.702	0.947	**15**	ResNet18	100	0.954	0.840
15	ResNet18	100	0.709	0.947	**14**	ResNet18	50	0.961	0.841
14	ResNet18	50	0.700	0.947	**1**	ResNet18	100	0.957	0.843
16	ResNet18	100	0.703	0.947	**8**	ResNet18	50	0.962	0.846
17	CNN	100	0.732	0.947	**12**	ResNet18	5	0.958	0.846
8	ResNet18	50	0.699	0.947	**17**	CNN	100	0.957	0.856
11	ResNet18	5	0.658	0.947	**3**	ResNet50	100	0.949	0.857
5	CNN	5	0.632	0.947	**4**	ResNet50	100	0.968	0.857
6	ResNet18	5	0.682	0.947	**11**	ResNet18	5	0.961	0.857
7	ResNet50	5	0.615	0.947	**16**	ResNet18	100	0.967	0.857
10	ResNet18	5	0.649	0.947	**5**	CNN	5	0.950	0.859
12	ResNet18	5	0.684	0.947	**2**	ResNet18	100	0.970	0.861
13	ResNet18	5	0.681	0.947	**18**	CNN	50	0.968	0.880
9	ResNet50	50	0.704	0.948	**20**	CNN	100	0.972	0.882
2	ResNet18	100	0.713	0.948	**7**	ResNet50	50	0.953	0.883
19	ResNet18	100	0.703	0.948	**19**	ResNet18	100	0.971	0.885

AUC = área sobre a curva (*Area under curve*); ACC = Exatidão (*Accuracy*)

Nas classes DermaMNIST e RetinaMNIST os resultados de ACC obtidos, para as configurações 19 e 20 (Tabela 10), foram novamente dos mais relevantes. Para a classe DermaMNIST, os resultados obtidos com a configuração 14 ganharam relevância, o que não ocorreu para mais nenhuma das classes estudadas (Tabela 8-Tabela **12**). É, ainda, importante realçar que, para esta mesma classe, a maioria dos resultados de ACC obtidos, para as configurações desenvolvidas nesta dissertação, são melhores do que os obtidos com as configurações previamente reportadas [237]. Para a classe RetinaMNIST, os melhores valores de ACC estão de acordo com os obtidos com a configuração 4.

Tabela 10. Resultados obtidos no processo de aprendizagem com as arquiteturas desenvolvidas, para as classes DermaMNIST e RetinaMNIST do dataset MedMNIST

DermaMNIST					RetinaMNIST				
Configuração	**Modelo**	***Epochs***	**AUC**	**ACC**	**Configuração**	**Modelo**	***Epochs***	**AUC**	**ACC**
5	CNN	5	0.847	0.680	**11**	ResNet18	5	0.601	0.230
16	ResNet18	100	0.900	0.707	**10**	ResNet18	5	0.628	0.273
3	ResNet50	100	0.886	0.710	**5**	CNN	5	0.703	0.435
7	ResNet50	5	0.855	0.710	**7**	ResNet50	5	0.624	0.453
8	ResNet18	50	0.892	0.716	**3**	ResNet50	100	0.719	0.490
12	ResNet18	5	0.881	0.717	**17**	CNN	100	0.713	0.510
4	ResNet50	100	0.895	0.719	**1**	ResNet18	100	0.727	0.515
11	ResNet18	5	0.887	0.720	**9**	ResNet50	50	0.688	0.515
17	CNN	100	0.913	0.720	**14**	ResNet18	50	0.724	0.518
1	ResNet18	100	0.899	0.721	**6**	ResNet18	5	0.736	0.523
10	ResNet18	5	0.881	0.723	**16**	ResNet18	100	0.719	0.525
6	ResNet18	5	0.894	0.724	**13**	ResNet18	5	0.711	0.535
2	ResNet18	100	0.896	0.727	**15**	ResNet18	100	0.720	0.543
15	ResNet18	100	0.899	0.729	**2**	ResNet18	100	0.721	0.543
9	ResNet50	50	0.886	0.731	**18**	CNN	50	0.715	0.551
13	ResNet18	5	0.899	0.733	**8**	ResNet18	50	0.725	0.553
18	CNN	50	0.900	0.739	**12**	ResNet18	5	0.720	0.553
20	CNN	100	0.915	0.740	**20**	CNN	100	0.731	0.553
14	ResNet18	50	0.899	0.740	**19**	ResNet18	100	0.726	0.554
19	ResNet18	100	0.913	0.742	**4**	ResNet50	100	0.717	0.555

AUC = área sobre a curva (*Area under curve*); ACC = Exatidão (*Accuracy*)

Para a classes BreastMNIST e OrganMNISTAxial são destacados os resultados de ACC obtidos para as configurações 18, 19 e 20 (Tabela 11). Para ambas as classes, estas configurações apresentaram valores relevantes e de acordo com a literatura [237], nomeadamente, os resultados obtidos com a configuração 2 para a classe BreastMNIST, e as configurações 2 e 4 para a classe OrganMNISTAxial.

Tabela 11. Resultados obtidos no processo de aprendizagem com as arquiteturas desenvolvidas, para as classes BreastMNIST e OrganMNISTAxial do dataset MedMNIST

BreastMNIST					OrganMNISTAxial				
Configuração	**Modelo**	***Epochs***	**AUC**	**ACC**	**Configuração**	**Modelo**	***Epochs***	**AUC**	**ACC**
12	ResNet18	5	0.605	0.269	**5**	CNN	5	0.984	0.841
10	ResNet18	5	0.651	0.712	**17**	CNN	100	0.991	0.888
5	CNN	5	0.681	0.731	**7**	ResNet50	5	0.992	0.893
6	ResNet18	5	0.648	0.731	**11**	ResNet18	5	0.994	0.899
7	ResNet50	5	0.527	0.731	**10**	ResNet18	5	0.993	0.901
11	ResNet18	5	0.629	0.731	**12**	ResNet18	5	0.994	0.906
13	ResNet18	5	0.666	0.731	**6**	ResNet18	5	0.995	0.912
9	ResNet50	50	0.866	0.821	**9**	ResNet50	50	0.995	0.914
4	ResNet50	100	0.863	0.833	**13**	ResNet18	5	0.995	0.915
8	ResNet18	50	0.881	0.833	**3**	ResNet50	100	0.995	0.916
15	ResNet18	100	0.903	0.833	**16**	ResNet18	100	0.996	0.920
14	ResNet18	50	0.900	0.840	**15**	ResNet18	100	0.995	0.921
16	ResNet18	100	0.888	0.846	**1**	ResNet18	100	0.995	0.921
3	ResNet50	100	0.879	0.853	**8**	ResNet18	50	0.996	0.923
17	CNN	100	0.878	0.859	**14**	ResNet18	50	0.995	0.925
1	ResNet18	100	0.897	0.859	**18**	CNN	50	0.993	0.926
18	CNN	50	0.909	0.872	**20**	CNN	100	0.996	0.929
20	CNN	100	0.911	0.872	**19**	ResNet18	100	0.995	0.931
19	ResNet18	100	0.915	0.874	**2**	ResNet18	100	0.997	0.931
2	ResNet18	100	0.915	0.878	**4**	ResNet50	100	0.997	0.931

AUC = área sobre a curva (*Area under curve*); ACC = Exatidão (*Accuracy*)

Tal como nos resultados apresentados para as classes anteriores (Tabela 11), para as classes OrganMNISTCoronal e OrganMNISTSagittal (Tabela 12), as configurações com os resultados mais relevantes são as configurações 18,19 e 20. Tendo em consideração os resultados obtidos com as configurações da literatura [237], uma vez mais, os valores obtidos de ACC são equivalentes aos obtidos para uma das classificações (configuração 2) e superiores às restantes configurações (1, 3 e 4).

Tabela 12. Resultados obtidos no processo de aprendizagem com as arquiteturas desenvolvidas, para as classes OrganMNISTCoronal e OrganMNISTSagittal do dataset MedMNIST.

OrganMNISTCoronal					OrganMNISTSagittal				
Configuração	**Modelo**	***Epochs***	**AUC**	**ACC**	**Configuração**	**Modelo**	***Epochs***	**AUC**	**ACC**
5	CNN	5	0.965	0.747	**5**	CNN	5	0.944	0.649
11	ResNet18	5	0.986	0.856	**11**	ResNet18	5	0.966	0.719
10	ResNet18	5	0.985	0.858	**10**	ResNet18	5	0.966	0.723
12	ResNet18	5	0.987	0.867	**12**	ResNet18	5	0.970	0.725
17	CNN	100	0.987	0.871	**17**	CNN	100	0.967	0.731
7	ResNet50	5	0.985	0.875	**7**	ResNet50	5	0.969	0.737
6	ResNet18	5	0.989	0.882	**14**	ResNet18	50	0.969	0.739
13	ResNet18	5	0.989	0.888	**8**	ResNet18	50	0.969	0.739
1	ResNet18	100	0.990	0.889	**6**	ResNet18	5	0.967	0.741
9	ResNet50	50	0.989	0.890	**3**	ResNet50	100	0.968	0.746
3	ResNet50	100	0.990	0.893	**16**	ResNet18	100	0.969	0.749
14	ResNet18	50	0.990	0.898	**9**	ResNet50	50	0.968	0.752
4	ResNet50	100	0.992	0.898	**15**	ResNet18	100	0.969	0.755
16	ResNet18	100	0.990	0.899	**13**	ResNet18	5	0.969	0.756
8	ResNet18	50	0.991	0.900	**1**	ResNet18	100	0.967	0.762
15	ResNet18	100	0.990	0.902	**4**	ResNet50	100	0.970	0.770
18	CNN	50	0.991	0.904	**18**	CNN	50	0.970	0.770
2	ResNet18	100	0.992	0.905	**20**	CNN	100	0.974	0.771
19	ResNet18	100	0.991	0.905	**19**	ResNet18	100	0.975	0.773
20	CNN	100	0.990	0.906	**2**	ResNet18	100	0.974	0.777

AUC = área sobre a curva (*Area under curve*); ACC = Exatidão (*Accuracy*)

Como se observa pela análise dos conjuntos de resultados obtidos para as diferentes classes (Tabela 8-Tabela **12**), de um modo geral, as configurações sujeitas à transformação 2 (Tabela 7), configurações 11 e 12, apresentam valores mais baixos de ACC. Estas duas configurações variam entre si apenas no *Batch*, com valores de 256 e 128, respetivamente. Este resultado sugere que a alteração deste parâmetro, por si só, não leva a obtenção de melhores resultados. Assim, e tendo em consideração que o tempo do processo de aprendizagem é bastante superior quando aplicado um *Batch* de 256, nas restantes configurações foi aplicado o *Batch* de 128.

Quando comparadas as configurações 11 e 12 com as configurações homólogas 10 e 6, respetivamente, em que apenas diferem na transformação (transformação 2), verifica-se que na maior parte dos estudos a transformação 2 permite obter melhores valores de ACC. Contudo, para estes 4 tipos de configuração, os valores obtidos de ACC são baixos. De facto, o número de iterações a que estas configurações estão sujeitas é reduzido (apenas 5), pelo que este resultado pode sugerir a importância do valor de *Epochs*.

Analisando as configurações 5 e 7, que apenas diferem na arquitetura em relação à configuração 6, constata-se que a configuração 5, à qual é aplicada a arquitetura *CNN*, apresenta os valores mais

baixos de ACC, exceto para a classe PneumoniaMNIST. Comparando os resultados obtidos aplicando a arquitetura *ResNet18* (configuração 6) e *ResNet50* (configuração 7), não é observado a prevalência de melhores resultados de uma arquitetura em relação à outra, uma vez que estas apresentam 60% e 40%, respetivamente, de melhores resultados entre si, nas classes estudadas. No entanto, uma vez mais, para estes tipos de configuração, os valores obtidos de ACC são baixos reforçando que este resultado pode estar associado ao baixo número de iterações.

Analisando a última configuração com o valor de *Epochs* de 5 (configuração 13) submetida à transformação 3, com um *Batch* de 128, e onde foi utilizada a arquitetura *ResNet18*, observa-se que, de um modo geral, esta configuração traduz-se em melhores resultados de ACC comparando com todas as homólogas anteriormente discutidas.

Analisando as configurações 8 e 9, homólogas das configurações 6 (com a arquitetura *ResNet18*) e 7 (com a arquitetura *ResNet50*) respetivamente, onde foi implementado um incremento de 10 vezes mais de iterações, verifica-se que, tal como sugerido anteriormente, o aumento do número de iterações reflete-se em um aumento do valor de ACC, sendo as configurações 8 e 9 geralmente mais interessantes. Contudo, uma vez mais, não é possível referir qual das arquiteturas é mais promissora pois, nas classes estudadas, a configuração 8, tal como a 9, apresenta 50% de melhores resultados nas classes estudadas.

As configurações 16 e 17 foram as últimas estudadas com a transformação 2. A diferença entre estas foi a arquitetura utilizada, *ResNet18* e *CNN*, respetivamente. Uma vez mais, a configuração onde foi aplicada a arquitetura *ResNet18*, configuração 16, foi a que permitiu obter melhores resultado de ACC, exceto para as classes DermaMNIST e BreastMNIST.

Curiosamente, quando foi aplicada a transformação 3, que apenas difere da transformação 2 no parâmetro de média da função de normalização, os resultados obtidos não foram promissores. Portanto, foi elaborada a transformação 4, novamente com o valor do parâmetro de média da função de normalização da transformação 2 (valor de 0.5).

Foram, assim, implementadas as configurações 18, 19 e 20 que, pela análise dos resultados se verifica apresentarem os melhores resultados, exceto para a classe ChestMNIST, sendo estes valores “semelhantes” entre si e, ainda, valores dentro da mesma gama ou melhores que os valores previamente reportando por Yang *et al*. (2020) [237]. É importante referir que estes resultados foram obtidos com uma resolução de imagem de 28x28 enquanto as configurações reportadas por Yang *et al.* (2020) [237] utilizaram resoluções de imagem que variavam entre 28x28 e 224x224. Os resultados obtidos para as configurações 18 e 20, que apenas diferem no número de iterações (a configuração 20 tem duas vezes mais iterações que a 18), reforçam, uma mais uma vez, que o aumento do número de iterações está relacionado com o aumento de valores de ACC. Por sua vez, as configurações 19 e 20, que apenas diferem no modelo implementado, destacam-se das restantes configurações onde foram aplicados os mesmos modelos, nas transformações utilizadas. Assim, são realçadas as combinações das transformações número 4 (Tabela 7) desenvolvidas no trabalho.

Adicionalmente às medidas de desempenho apresentadas, foi tido em consideração a duração do processo de treino para cada configuração. Uma vez mais, comparando as configurações com os melhores resultados, 19 e 20, é importante referir que o processo de treino da configuração 19, que contempla a *ResNet18*, demora, em média, o dobro do tempo do processo de treino da configuração 20, que utiliza a *CNN*.

Em suma, algumas conclusões podem ser inferidas para a classificação de imagens do dataset *MedMNIST*:

(i) Os resultados variam em função da classe, o que era espectável, uma vez que este dataset é um conjunto de vários dataset (classes) com propriedades de imagem distintas;

(ii) Para este tipo de dataset, o melhor procedimento a adotar será implementar para cada classe um conjunto de configurações /modelo específico;

(iii) O número de iterações tem influência no processo de aprendizagem (considerando a variação do número de iterações estudado);

(iv) O parâmetro *Batch,* por si só, não influencia os resultados;

(v) Não foram detetadas diferenças significativas de desempenho entre a arquitetura *ResNet18* e *ResNet50*, sendo a arquitetura *CNN* menos eficiente que as anteriormente referidas;

(vi) É importante ter em consideração o fator “tempo de aprendizagem” quando comparados os diferentes resultados;

(vii) As transformações desenvolvidas neste trabalho são promissoras, pela seguinte ordem: 4 > 2 > 3 > 1.

4.1.2 Resultados de classificação obtidos com o Dataset MedNIST

Para a classificação de imagens médicas, com o conjunto de dados *MedNIST*, foram implementadas diferentes arquiteturas, nomeadamente, *EfficientNet-B0, VGG16*, *DensetNet121*, *DenseNet169*, *ResNet18* e *CNN*. Este dataset é constituído por seis classes (Figura 42).

As métricas aplicadas na avaliação do desempenho em cada um dos modelos implementados foram a precisão, exatidão (ACC), *Recall* e a F1-*Score.* Estas foram selecionadas por serem as mais indicadas para a tarefa de classificação deste tipo de conjuntos de dados [238]. Relativamente à função usada no cálculo das perdas, foi implementada a entropia cruzada [239].

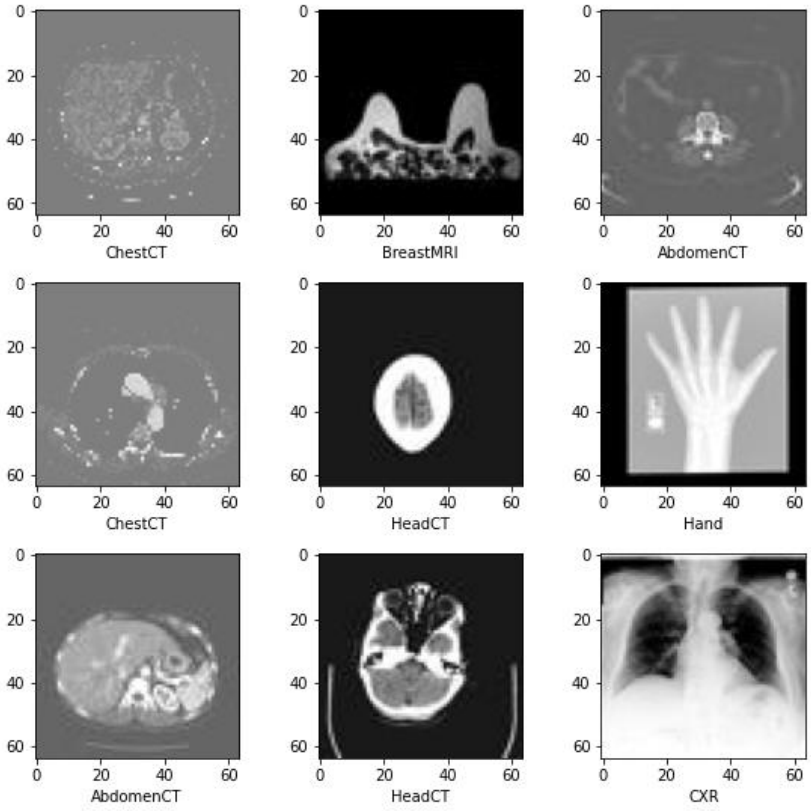


Figura 42. Exemplo de uma imagem para cada classe do Dataset *MedNIST*.

Na Tabela 13 estão sumarizados os resultados obtidos após o processo de aprendizagem com as diferentes arquiteturas, para cada uma das classes do dataset *MedNIST*. Na Figura 43 está presente, a título ilustrativo, um conjunto de imagens corretamente classificadas após o processo de treino com a arquitetura *CNN*.

Tabela 13. Resultados obtidos no processo de aprendizagem para classificação de imagens do dataset *MedNIST*, para as arquiteturas desenvolvidas.

EfficientNet-B0					**VGG16**				
Classe	**Precisão**	***Recall***	**F1-score**	**ACC**	**Classe**	**Precisão**	***Recall***	**F1-score**	**ACC**
AbdomenCT	1.0000	1.0000	1.0000	1.0000	**AbdomenCT**	0.9974	0.9760	0.9866	0.9953
BreastMRI	1.0000	1.0000	1.0000		**BreastMRI**	1.0000	1.0000	1.0000	
CXR	1.0000	1.0000	1.0000		**CXR**	1.0000	0.9990	0.9994	
ChestCT	1.0000	1.0000	1.0000		**ChestCT**	0.9765	1.0000	0.9881	
Hand	1.0000	1.0000	1.0000		**Hand**	0.9990	0.9995	0.9992	
HeadCT	1.0000	1.0000	1.0000		**HeadCT**	1.0000	0.9954	0.9954	
DenseNet121					**DenseNet169**				
Classe	**Precisão**	***Recall***	**F1-score**	**ACC**	**Classe**	**Precisão**	***Recall***	**F1-score**	**ACC**
AbdomenCT	0.9995	0.9995	0.9995	0.9997	**AbdomenCT**	1.0000	1.0000	1.0000	0.9997
BreastMRI	1.0000	1.0000	1.0000		**BreastMRI**	1.0000	1.0000	1.0000	
CXR	1.0000	0.9995	0.9998		**CXR**	1.0000	0.9985	0.9993	
ChestCT	0.9995	1.0000	0.9998		**ChestCT**	1.0000	1.0000	1.0000	
Hand	0.9995	1.0000	0.9997		**Hand**	0.9984	1.0000	0.9992	
HeadCT	1.0000	0.9995	0.9998		**HeadCT**	1.0000	1.0000	1.0000	
ResNet18					**CNN**				
Classe	**Precisão**	***Recall***	**F1-score**	**ACC**	**Classe**	**Precisão**	***Recall***	**F1-score**	**ACC**
AbdomenCT	1.0000	0.9186	0.9576	0.9733	**AbdomenCT**	0.9962	0.9938	0.9950	0.9950
BreastMRI	0.9966	0.9989	0.9977		**BreastMRI**	0.9983	0.9989	0.9986	
CXR	0.9990	0.9824	0.9906		**CXR**	0.9960	0.9920	0.9940	
ChestCT	0.9775	0.9478	0.9624		**ChestCT**	0.9902	0.9985	0.9943	
Hand	0.9336	0.9964	0.9640		**Hand**	0.9915	0.9936	0.9926	
HeadCT	0.9427	0.9995	0.9702		**HeadCT**	0.9980	0.9936	0.9958	

ACC = Exatidão (*Accuracy*)

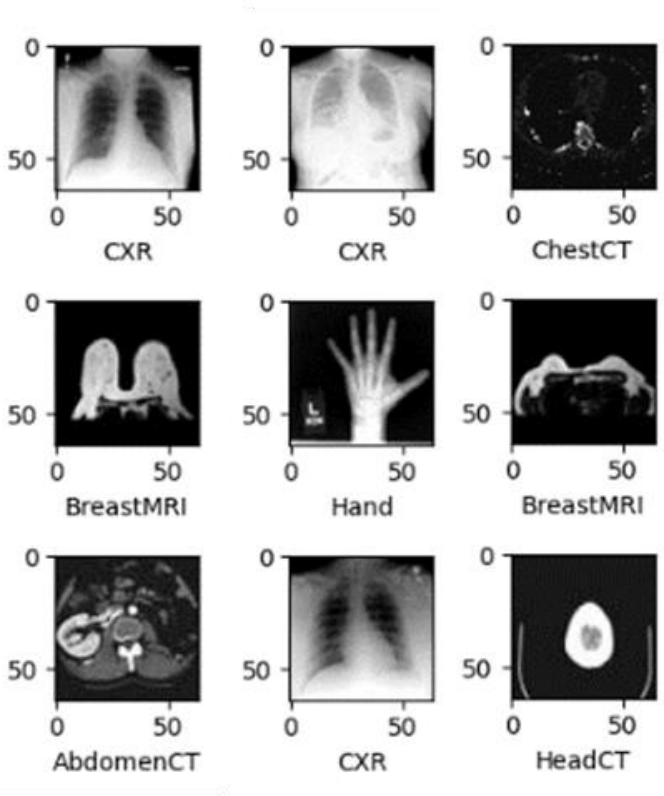


Figura 43. Exemplo de alguns resultados de classificação de imagens no dataset *MedNIST* obtidos com a arquitetura *CNN*.

Analisando os resultados presentes na Tabela 13, verifica-se que, de um modo geral, os valores de precisão obtidos para cada classe do dataset, nas várias arquiteturas implementadas, foram elevados (valores de precisão superiores a 0.9336). Contudo, verifica-se que os valores mais baixos de precisão são obtidos com a arquitetura *CNN* e a *ResNet18*, exceto para a classe AbdomenCT em que os valores mais baixos resultam da aprendizagem com as arquiteturas *VGG16* e *CNN*, e para a classe ChestCT em que os valores mais baixos resultam da aprendizagem com as arquiteturas *VGG16* e a *ResNet18*. De facto, os resultados de precisão obtidos com a arquitetura *VGG16* variam bastante em função da classe estudada, oscilando entre a que apresenta os maiores ou menores valores de precisão.

Comparando o valor de exatidão, para todo o dataset (no conjunto das seis classes), entre as diferentes arquiteturas comprova-se que a arquitetura *EfficientNet-B0* obteve o melhor resultado, sendo que as arquiteturas *CNN* e a *ResNet18* apresentaram os menores valores de exatidão, tal como de precisão.

Na Figura 44 é possível verificar os resultados de classificação de cada classe, estando presente a relação entre os valores reais e os valores previstos, por arquitetura implementada. Deste modo, é possível saber que troca de classificação de classe ocorreu. Por exemplo, para a arquitetura *VGG16* (Figura 44B) consta-se que, das 2000 amostras de teste da classe AbdomenCT, 1952 apresentaram uma classificação correta enquanto 48 amostras foram classificadas como sendo da classe ChestCT.

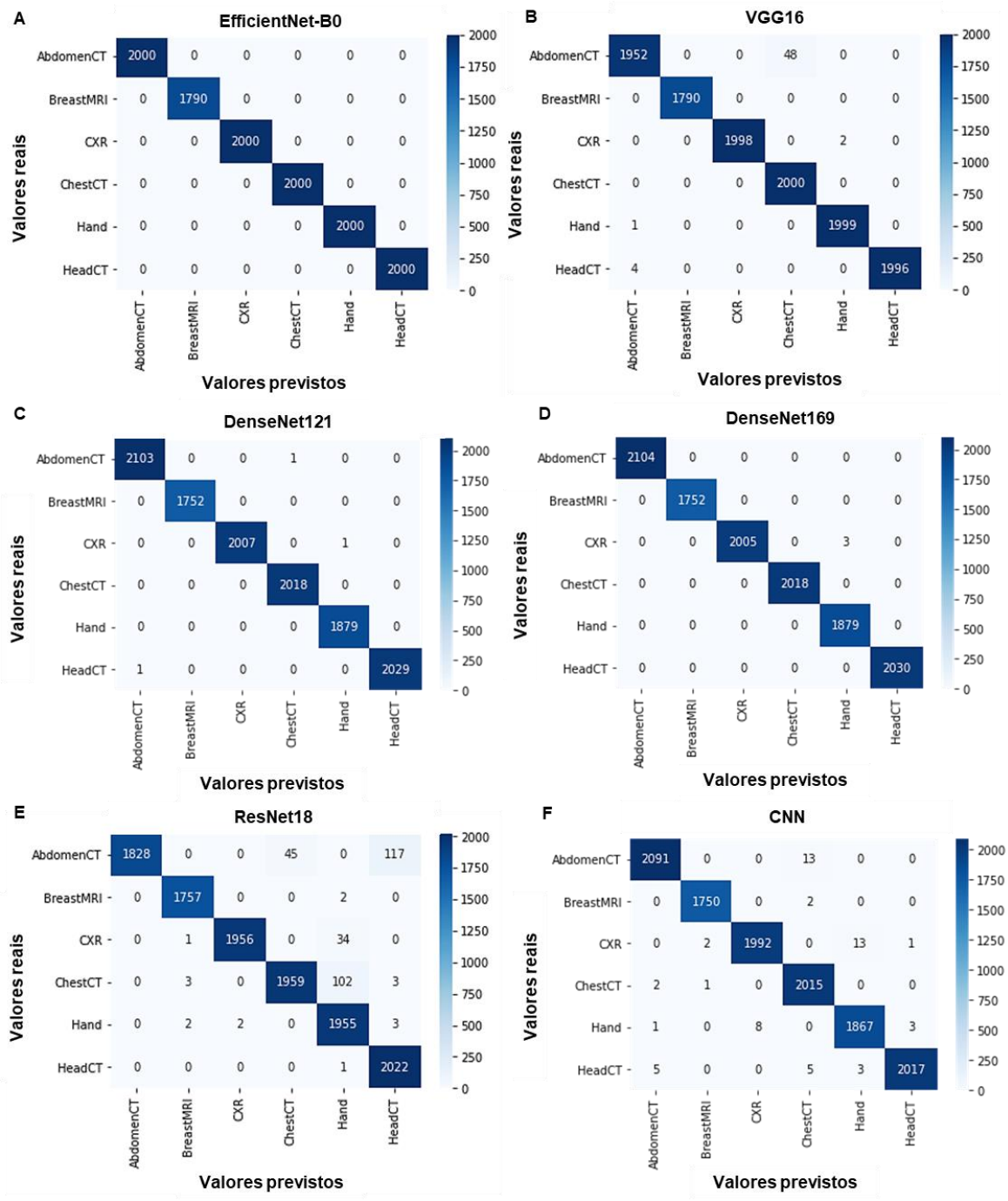


Figura 44. Matriz da relação dos resultados reais *vs* resultados previstos de acordo com as arquiteturas exploradas, nomeadamente: **A**) *EfficientNet-B0*: **B**) *VGG16*; **C**) *DensetNet121*; **D**) *DenseNet169*; **E**) *ResNet18*; **F**) *CNN.*

Pela análise dos gráficos dos valores de exatidão (ACC) em função do número de iterações (*Epochs*) (Figura 45A-F) e dos gráficos de perdas em função do número de iterações (*Epochs*) (Figura 45G-L) obtidos ao longo do processo de treino, verifica-se que no decorrer das iterações no processo de treino (*Train*) há um aumento do valor de ACC, sendo este aumento inversamente proporcional ao valor das perdas, tal como previsto. Um facto interessante observado para as arquiteturas *EfficientNet-B0, VGG16*, *DensetNet121* e *DenseNet169* é que o valor máximo de ACC é atingido para um número de iterações inferior a dez (Figura 45A-D), tal como o valor mínimo de

perdas (Figura 45G-J). O mesmo não se verifica para as arquiteturas *ResNet18* e *CNN*. No caso destas arquiteturas, o valor máximo de ACC é atingido para um número aproximado de 40 iterações (Figura 45E-F), assim como o valor mínimo de perdas (Figura 45K-L).

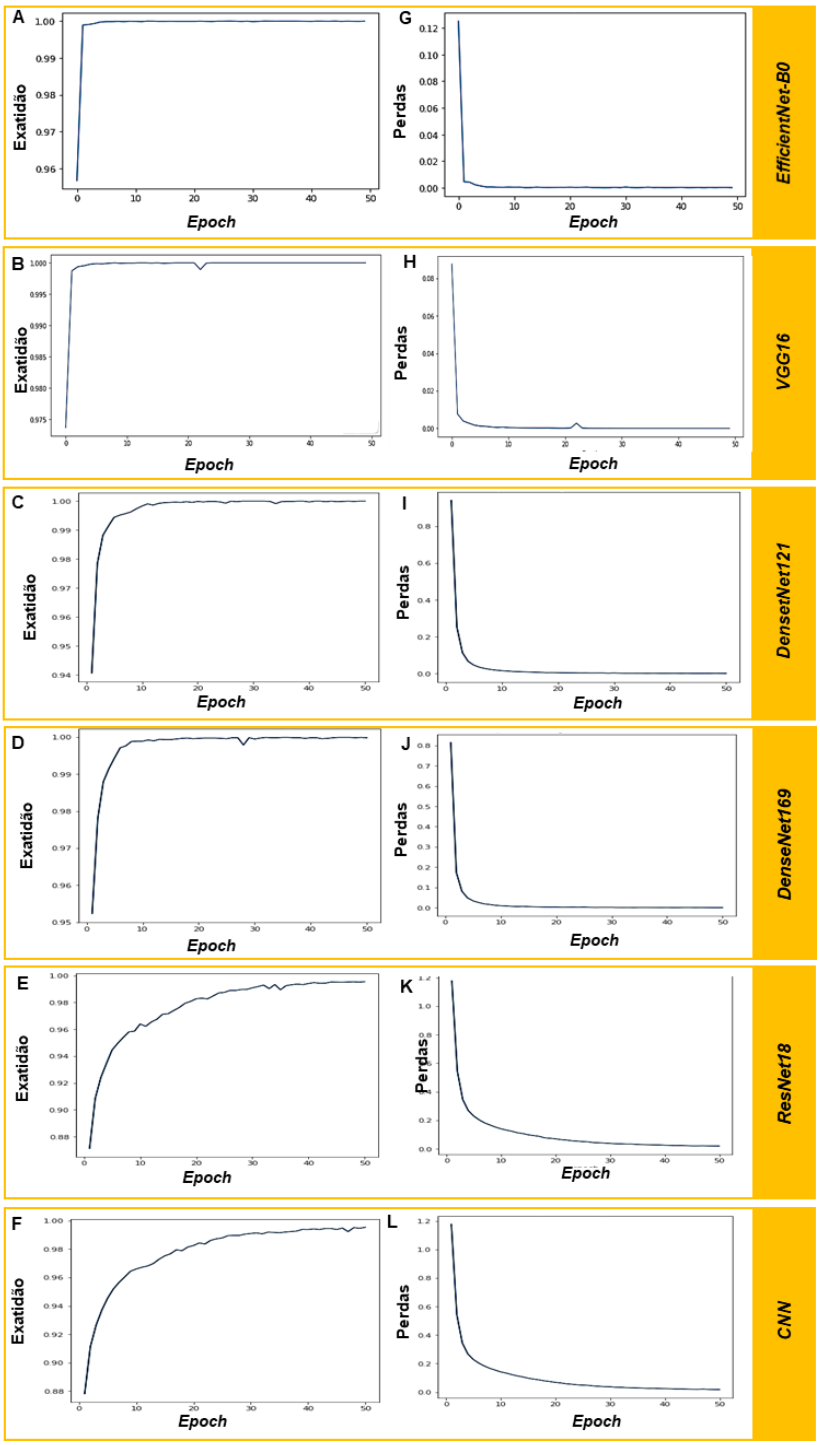


Figura 45. Gráficos das métricas obtidas no processo de aprendizagem para a tarefa de classificação de imagens do dataset *MedNIST*. **A**, **B**, **C**, **D**, **E**, **F**: Gráfico dos valores de exatidão (ACC) *vs* número de iterações (*Epochs*) obtidos com as arquiteturas *EfficientNet-B0, VGG16*, *DensetNet121*, *DenseNet169*, *ResNet18* e *CNN*, respetivamente. **G**, **H**, **I**, **J**, **K**, **L**: Gráfico dos valores de perdas *vs* número de iterações (*Epochs*) obtidos com as arquiteturas *EfficientNet-B0, VGG16*, *DensetNet121*, *DenseNet169*, *ResNet18* e *CNN*, respetivamente.

Em suma, algumas conclusões podem ser inferidas para a classificação de imagens do dataset *MedNIST*:

(i) As arquiteturas implementadas têm, tal como previsto, influência na classificação, sendo a *ResNet18* e a *CNN* as que apresentaram os resultados menos promissores;

(ii) Para este dataset a influência do número de iterações do processo de treino é menor para a maioria dos modelos implementados;

(iii) A classificação de imagens deste dataset é mais simples quando comparado com o dataset *MedMNIST*.

4.2 Segmentação de Imagens

Para a tarefa de deteção e segmentação de imagens foram selecionados os dataset *DSB2018* e o *ICPR2012*. Após a aplicação das metodologias de pré-processamento de imagem e implementadas as diferentes arquiteturas, os conjuntos de dados foram submetidos ao processo de treino através das configurações planeadas.

4.2.1 Resultados de segmentação obtidos com o Dataset *DSB2018*

Para a deteção e segmentação de imagens foi selecionado um dataset de imagens histológicas (*DSB2018*). O objetivo principal desta parte de investigação consistiu na deteção de núcleos celulares. Para a análise do dataset selecionado, *DSB2018*, foram implementadas quatro arquiteturas, nomeadamente, *CNN*, *U-NET*, *W-UNET* e a *UNET++*.

A implementação de cada uma das arquiteturas contempla duas fases: a fase de aprendizagem, isto é, a fase de treino e de validação; e a fase de teste. Na Tabela 14 estão presentes os valores das métricas resultantes da fase de teste, com cada uma das arquiteturas. De modo a complementar o estudo de segmentação de núcleos, estão presentes ainda as métricas referentes ao processo de treino e validação. Adicionalmente, na Tabela 14, estão presentes algumas métricas reportadas para este dataset na literatura com diferentes arquiteturas [198, 240].

Tabela 14. Resultados obtidos no processo de segmentação de imagem no dataset *DSB2018*, com as arquiteturas desenvolvidas.

Fase	Perdas DICE	Exatidão	Coeficiente DICE	IoU	*Recall*	Precisão	Modelo
Treino	0.1553	0.9566	0.8449	0.7324	0.8433	0.8454	
Validação	0.1531	0.9587	0.8459	0.7339	0.8640	0.8399	***CNN***
Teste	0.0161	1.000	0.9835	0.9675	1.0000	1.0000	
Treino	0.1670	0.9479	0.8326	0.7138	0.9446	0.7457	
Validação	0.1662	0.9459	0.845	0.7335	0.9411	0.7469	***UNET***
Teste	0.0084	1.0000	0.9916	0.9832	1.0000	1.0000	
Treino	0.0542	0.9850	0.9459	0.8974	0.9564	0.9369	
Validação	0.0960	0.9738	0.9104	0.8363	0.9151	0.8970	***W_UNET***
Teste	0.0079	1.0000	0.9921	0.9843	1.0000	1.0000	
Treino	0.0907	0.9744	0.9094	0.8341	0.9249	0.8954	
Validação	0.1116	0.9689	0.8918	0.8052	0.9148	0.8647	***UNET++***
Teste	0.0159	1.0000	0.9848	0.9701	1.0000	1.0000	
N.R.	-	-	-	0.9077	-	-	***U-NET*** **[198]**
	-	-	-	0.9263	-	-	***UNET++*** **[198]**
	-	-	-	0.9092	-	-	***Wide UNET*** **[198]**
	-	-	0.9215	-	-	-	***R2U-NET*** **[240]**

N.R. Não referido

Comparando os resultados obtidos na fase de teste com os da fase de aprendizagem, estes melhoram consideravelmente em todas as métricas, para todas as arquiteturas implementadas. Comparando as arquiteturas entre si, todas apresentam o mesmo valor de exatidão e precisão. No entanto, estas métricas não são as mais adequadas a ter em consideração para este tipo de tarefa. Assim, analisando as métricas mais utilizadas para este estudo, o IoU e o coeficiente DICE, verifica-se que os melhores resultados são obtidos com as arquiteturas *U-NET* e *W-UNET*, seguindo-se a arquitetura *UNET*++ e, por fim, a *CNN*. Contudo, é importante realçar que, os valores obtidos de IoU (0.9675 - 0.9843) são superiores aos reportados na literatura (0.9077 - 0.9263) [198]. O mesmo se verifica quando comparados os valores de coeficiente de DICE obtidos (0.9835 - 0.9916) com o valor reportado (0.9215) [240], reforçando assim que os resultados obtidos são bastante promissores. É, ainda, importante referir que é com a arquitetura *CNN* que o processamento é mais rápido, para um mesmo número de iterações.

Nas Figura 46 e Figura 47 estão presentes os gráficos das diferentes métricas aplicadas, em função do número de iterações (*Epochs*) na fase de aprendizagem. Focando nos gráficos das métricas mais relevantes para este estudo, o IoU (Figura 46D,J e Figura 47D,J) e o coeficiente DICE (Figura 46C,I e Figura 47C,I), verifica-se que no decorrer das iterações há um aumento do valor de IoU e do coeficiente de DICE. O valor máximo de IoU é atingido para um número de iterações de aproximadamente 30, 40, 35 e 40, para as arquiteturas *CNN*, *U-NET*, *W-UNET* e a *UNET*++, respetivamente. O valor máximo de coeficiente de DICE é atingido para um número de iterações de aproximadamente 20, 25, 30 e 20, para as arquiteturas *CNN*, *U-NET*, *W-UNET* e a *UNET*++, respetivamente.

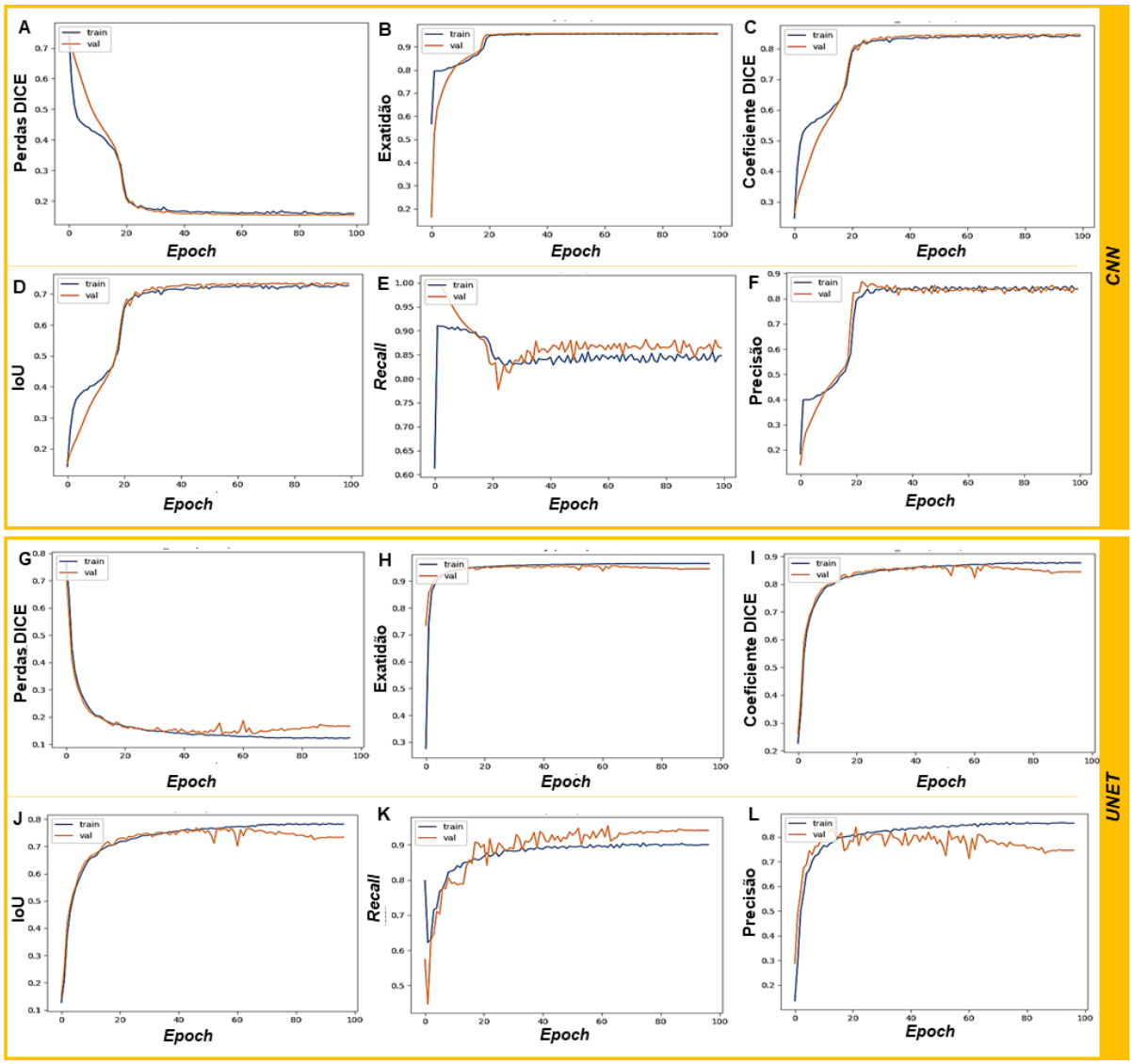


Figura 46. Gráficos das métricas obtidas na fase de aprendizagem na deteção e segmentação de imagens para as arquiteturas *CNN* e *UNET*. **A, G**: Gráfico dos valores de perdas *vs* número de iterações (*Epochs*) obtidos com as arquiteturas *CNN e UNET*, respetivamente; **B, H**: Gráfico dos valores de exatidão *vs* número de iterações (*Epochs*) obtidos com as arquiteturas *CNN e UNET*, respetivamente; **C, I**: Gráfico dos valores de coeficiente DICE *vs* número de iterações (*Epochs*) obtidos com as arquiteturas *CNN e UNET*, respetivamente; **D, J**: Gráfico dos valores de IoU *vs* número de iterações (*Epochs*) obtidos com as arquiteturas *CNN e UNET*, respetivamente; **E, K**: Gráfico dos valores de *recall vs* número de iterações (*Epochs*) obtidos com as arquiteturas *CNN e UNET*, respetivamente; **F, L**: Gráfico dos valores de precisão *vs* número de iterações (*Epochs*) obtidos com as arquiteturas *CNN e UNET*, respetivamente.

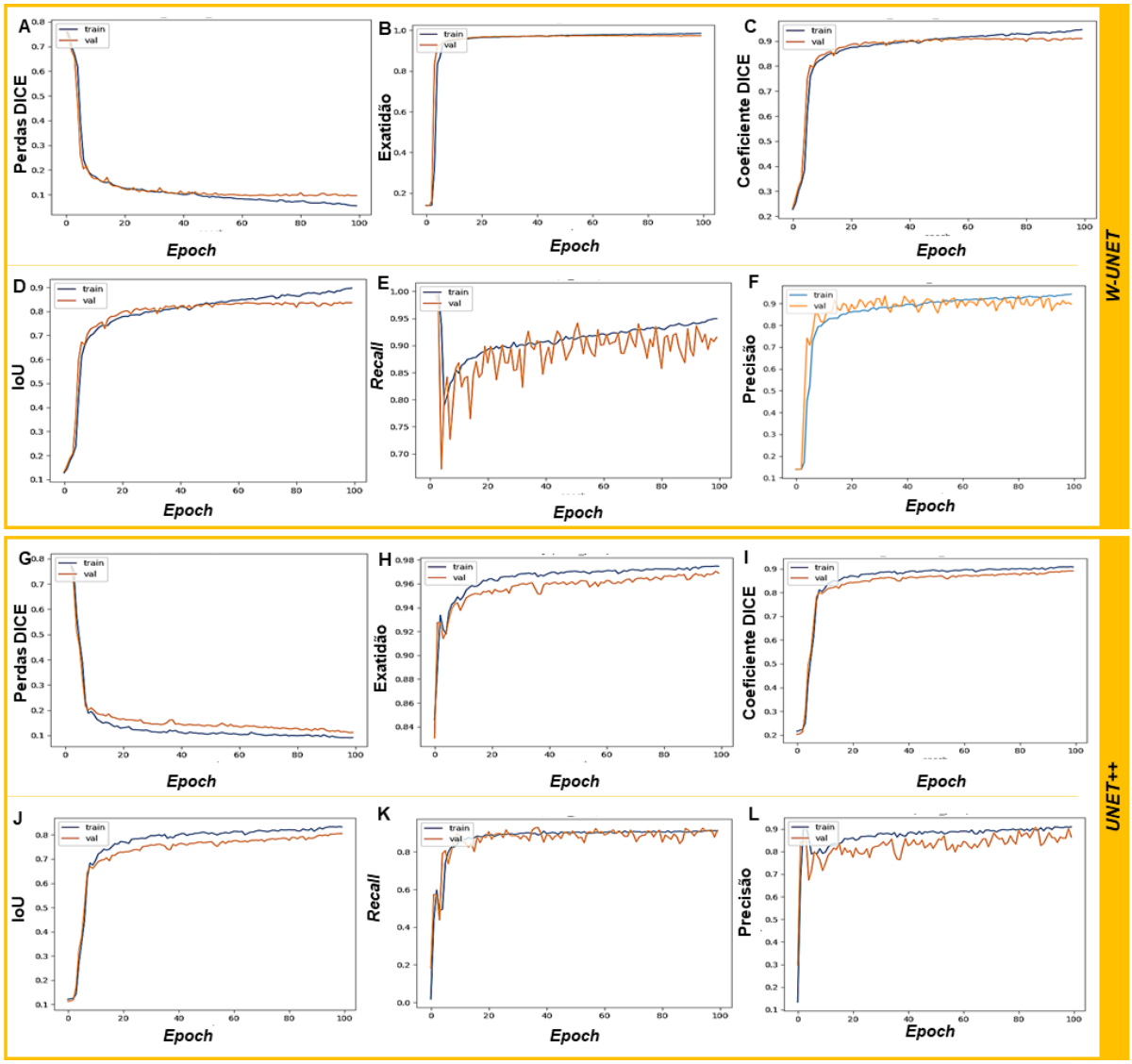


Figura 47.Gráficos das métricas obtidas na fase de aprendizagem na deteção e segmentação de imagens para as arquiteturas *W-UNET* e *UNET*++. **A, G**: Gráfico dos valores de perdas *vs* número de iterações (*Epochs*) obtidos com as arquiteturas *W-UNET* e *UNET*++, respetivamente; **B, H**: Gráfico dos valores de exatidão *vs* número de iterações (*Epochs*) obtidos com as arquiteturas *W-UNET* e *UNET*++, respetivamente; **C, I**: Gráfico dos valores de coeficiente DICE *vs* número de iterações (*Epochs*) obtidos com as arquiteturas *W-UNET* e *UNET*++, respetivamente; **D, J**: Gráfico dos valores de IoU *vs* número de iterações (*Epochs*) obtidos com as arquiteturas *W-UNET* e *UNET*++, respetivamente; **E, K**: Gráfico dos valores de *recall vs* número de iterações (*Epochs*) obtidos com as arquiteturas *W-UNET* e *UNET*++, respetivamente; **F, L**: Gráfico dos valores de precisão *vs* número de iterações (*Epochs*) obtidos com as arquiteturas *W-UNET* e *UNET*++, respetivamente.

Depois de concluído o processo de treino, o modelo foi testado com as imagens destinadas ao processo de treino e validação, e mais importante, com as imagens desconhecidas, ou seja, destinadas apenas ao teste. A escolha das imagens a serem segmentadas foi aleatória. Estes resultados estão presentes nas Figura 49 - Figura 52, de acordo com a arquitetura implementada.

Nos testes com os dados de treino e validação são apresentadas quatro informações: a imagem original do dataset, a respetiva máscara, a predição realizada com o modelo implementado e, ainda, uma figura de sobreposição das anteriores três imagens. Esta sobreposição é importante para a deteção da veracidade dos resultados. Para o processo de teste com as imagens desconhecidas (imagens de teste), são apresentadas a imagem original utilizada e a imagem resultante da predição realizada pelo modelo.

Focando na imagem da sobreposição é possível através da variação de cores interpretar a veracidade dos resultados obtidos na segmentação. A cor verde, roxo, vermelho e branco representam os verdadeiros positivos, os falsos positivos, os falsos negativos e os verdadeiros negativos, respetivamente (Figura 48).

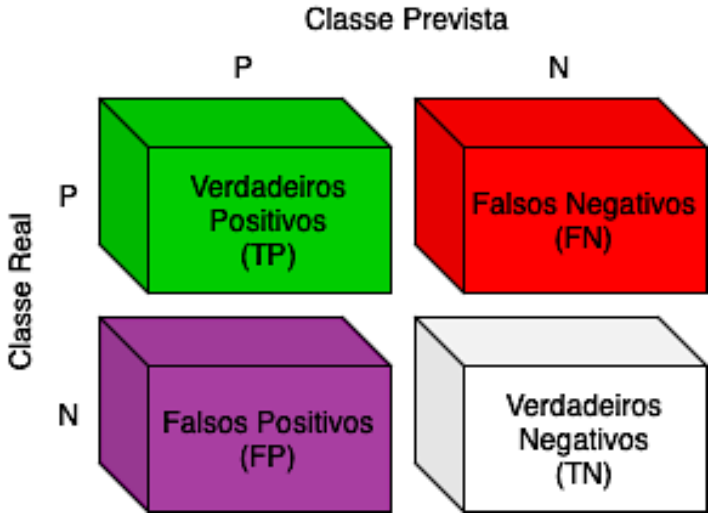


Figura 48. Matriz de confusão com o correspondente mapa de cores dos resultados de sobreposição.

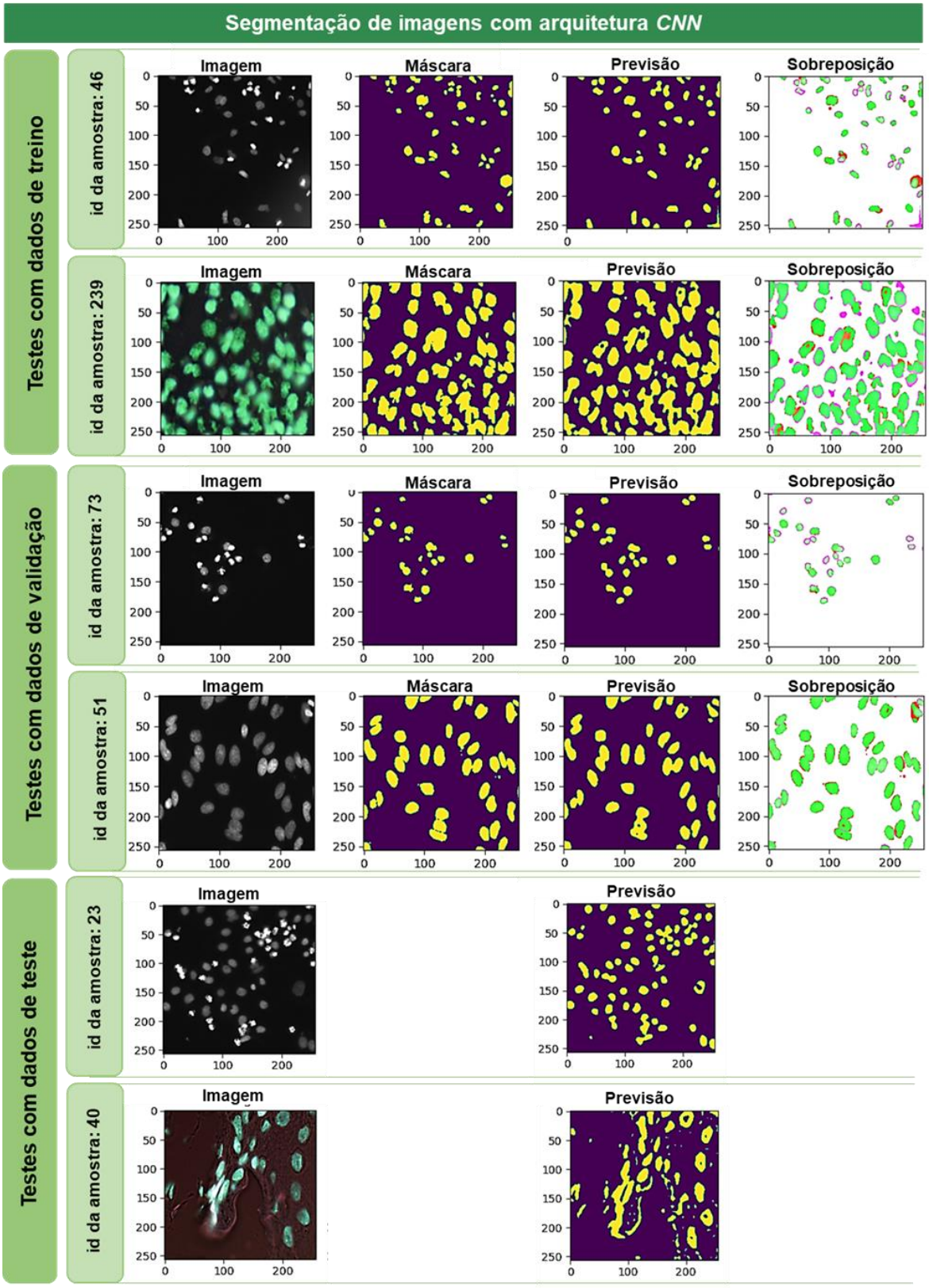


Figura 49. Imagens resultantes dos testes de segmentação obtidos com a arquitetura *CNN*, para o dataset *DSB2018*. A imagem, a máscara, a previsão e a sobreposição correspondem à imagem real do dataset, à anotação dos núcleos relativos à imagem, ao resultado obtido e à sobreposição das três imagens referidas anteriormente, respetivamente. As cores verde, roxo, vermelha e branco representam os núcleos, os verdadeiros positivos, os falsos positivos, os falsos negativos e os verdadeiros negativos, respetivamente.

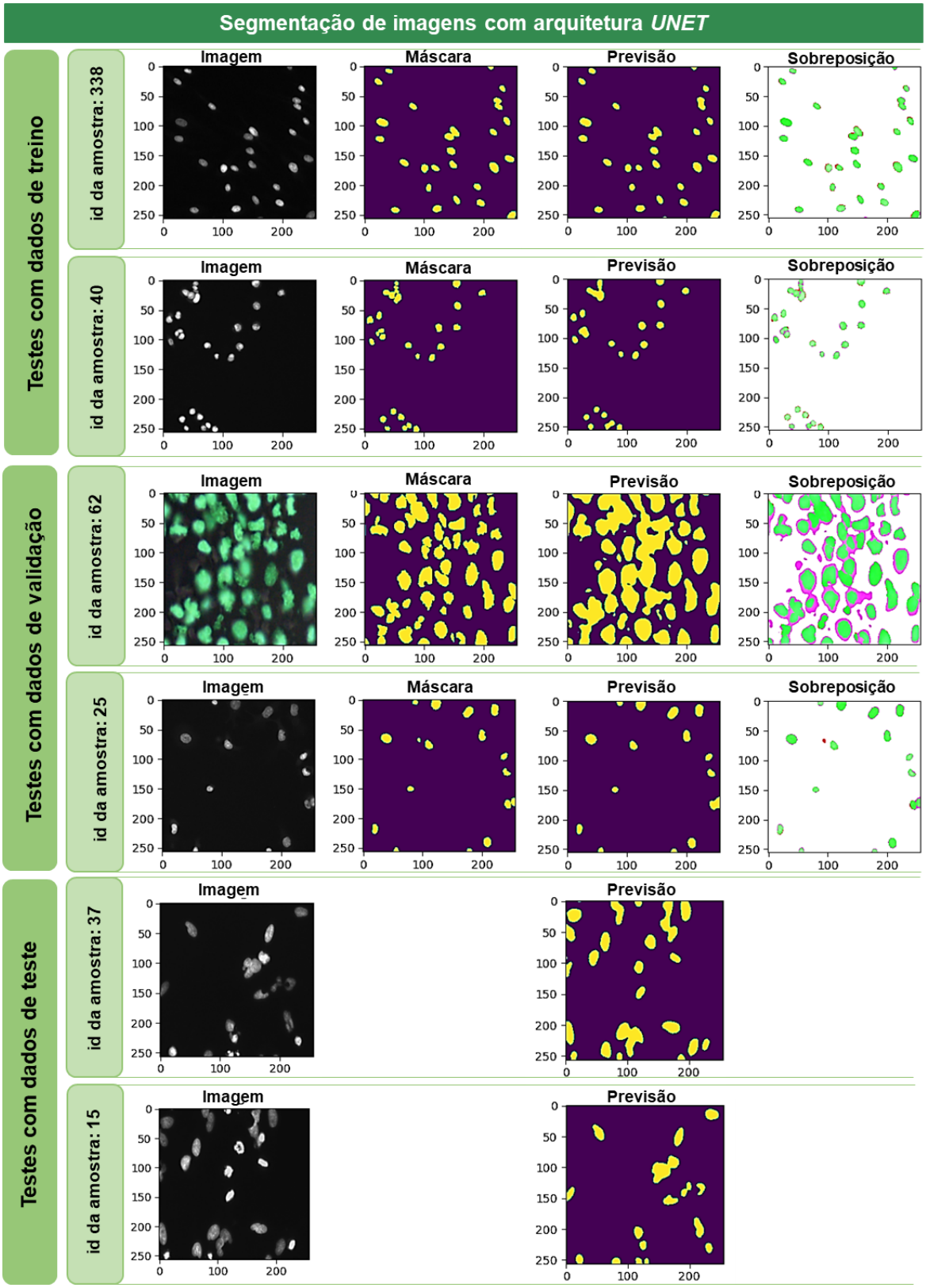


Figura 50. Imagens resultantes dos testes de segmentação obtidos com a arquitetura *U-NET*, para o dataset *DSB2018.* A imagem, a máscara, a previsão e a sobreposição correspondem à imagem real do dataset, à anotação dos núcleos relativos à imagem, ao resultado obtido e à sobreposição das três imagens referidas anteriormente, respetivamente. As cores verde, roxo, vermelha e branco representam os núcleos, os verdadeiros positivos, os falsos positivos, os falsos negativos e os verdadeiros negativos, respetivamente.

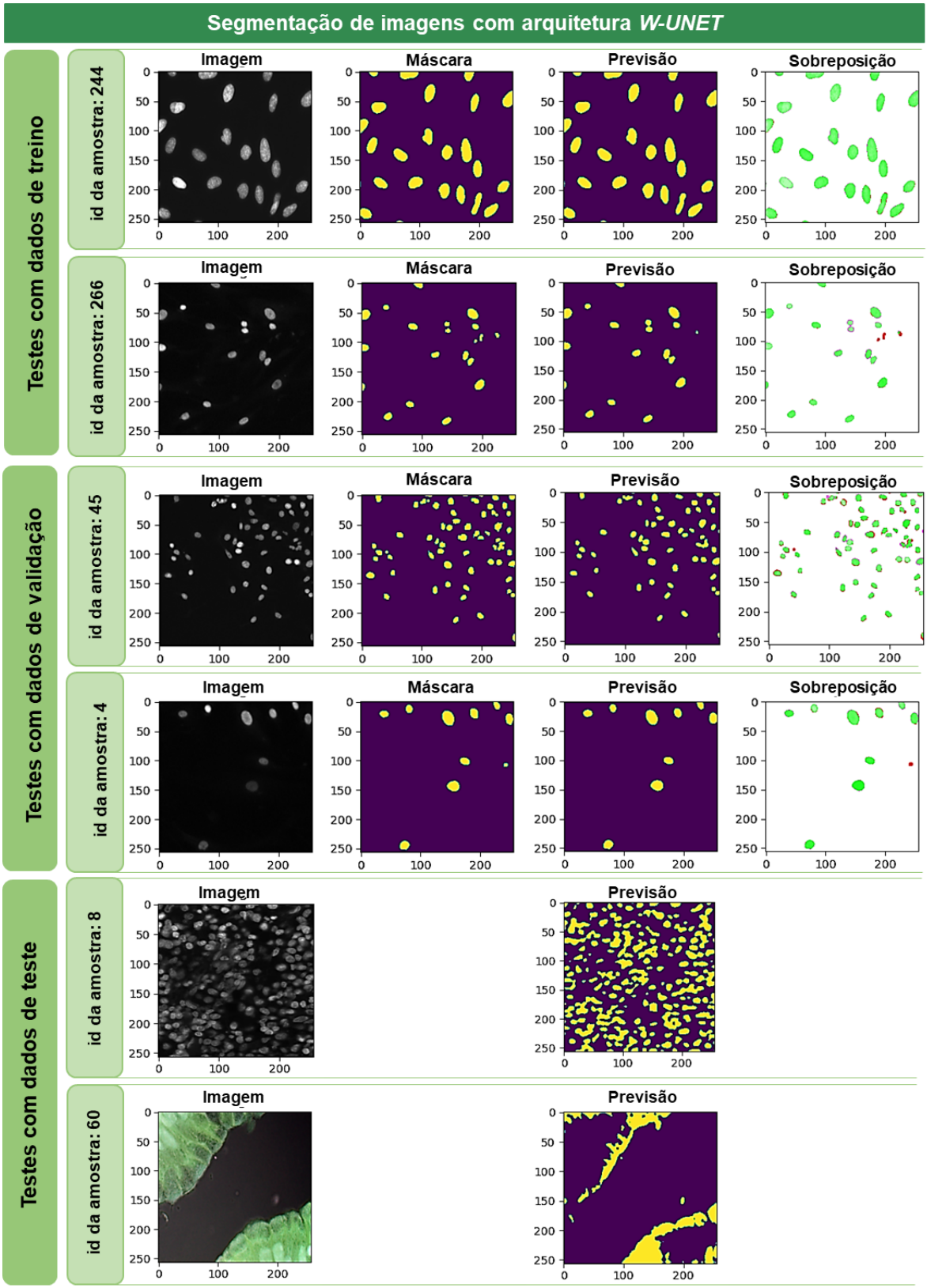


Figura 51. Imagens resultantes dos testes de segmentação obtidos com a arquitetura *W-UNET*, para o dataset *DSB2018.* A imagem, a máscara, a previsão e a sobreposição correspondem à imagem real do dataset, à anotação dos núcleos relativos à imagem, ao resultado obtido e à sobreposição das três imagens referidas anteriormente, respetivamente. As cores verde, roxo, vermelha e branco representam os núcleos, os verdadeiros positivos, os falsos positivos, os falsos negativos e os verdadeiros negativos, respetivamente.

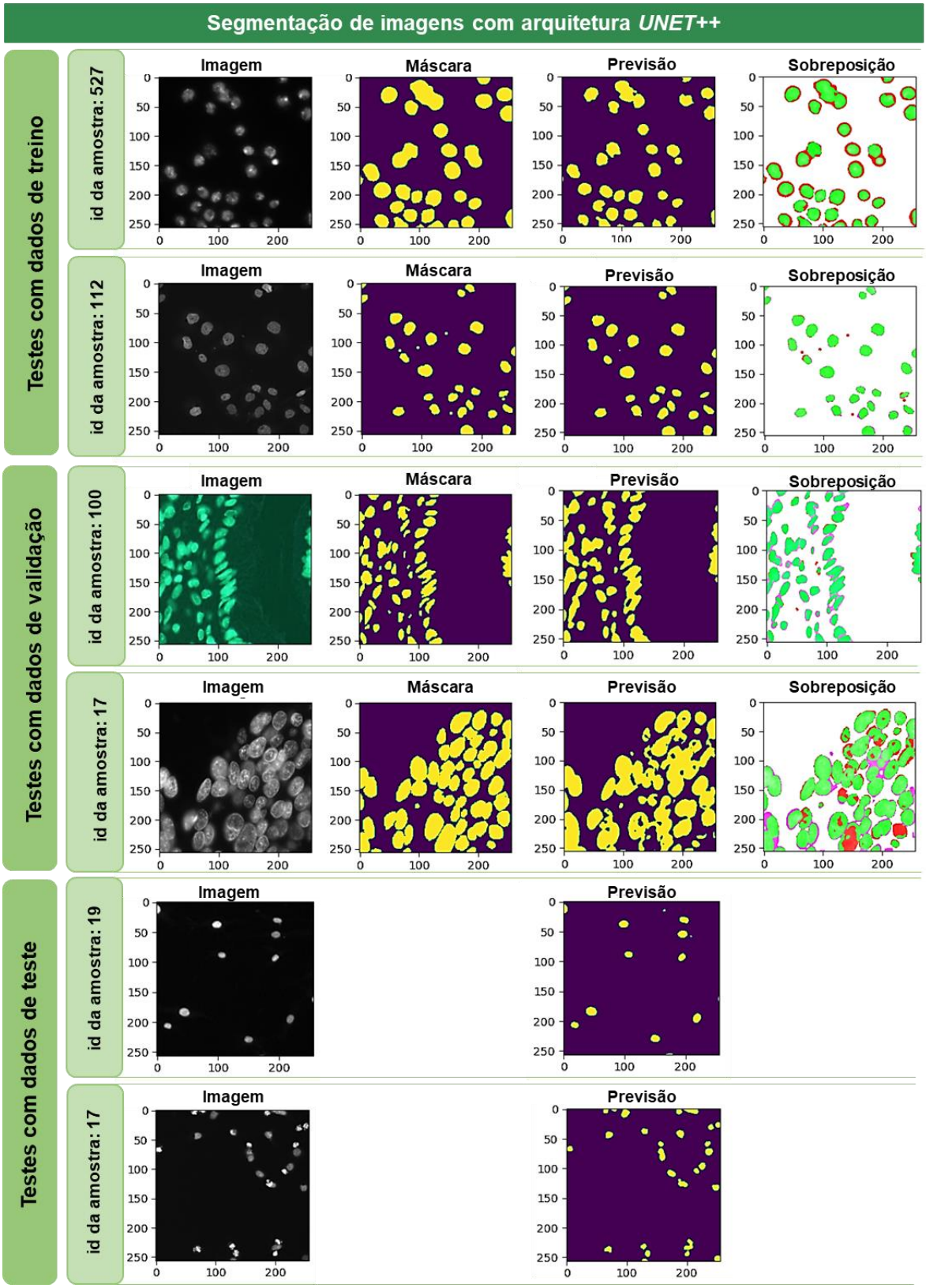


Figura 52. Imagens resultantes dos testes de segmentação obtidos com a arquitetura *UNET++*, para o dataset *DSB2018*. A imagem, a máscara, a previsão e a sobreposição correspondem à imagem real do dataset, à anotação dos núcleos relativos à imagem, ao resultado obtido e à sobreposição das três imagens referidas anteriormente, respetivamente. As cores verde, roxo, vermelha e branco representam os núcleos, os verdadeiros positivos, os falsos positivos, os falsos negativos e os verdadeiros negativos, respetivamente.

4.2.2 Resultados de segmentação obtidos com o Dataset *ICPR2012*

Uma das particularidades da segmentação de imagens de células, mais concretamente a identificação de núcleos celulares, é a possibilidade de identificar diferentes estados das mesmas. Um exemplo concreto é a identificação de células em fase de mitose. A mitose é um processo de divisão celular que apresenta grande importância para os organismos. Nos seres multicelulares, a mitose é importante para garantir o crescimento desses indivíduos e também para a regeneração dos tecidos. Contudo, em patologias como, por exemplo, o cancro, ocorrem anomalias no ciclo celular levando a um aumento descontrolado do número de mitoses. Assim, a identificação destas pode permitir um diagnóstico mais eficaz e precoce.

Para o estudo desenvolvido com o dataset *ICPR2012*, cujo objetivo consiste na segmentação e deteção de núcleos, mas com a particularidade de apenas identificar núcleos que se encontrassem em mitose, foi implementada a arquitetura *U-NET*.

A implementação da arquitetura contempla uma vez mais a fase de aprendizagem, isto é, a fase de treino e de validação, e a fase de teste. Na Figura 53 estão presentes resultados visuais obtidos com a arquitetura *U-NET* para o dataset *ICPR2012*.

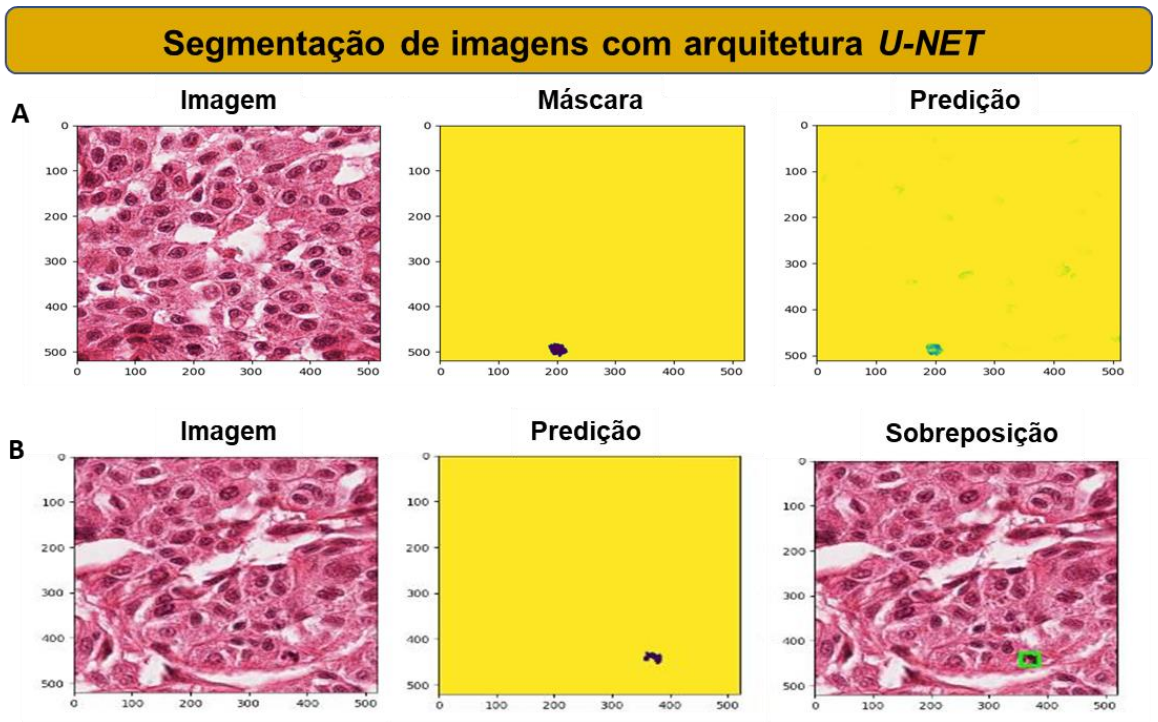


Figura 53. Imagens resultantes dos testes de segmentação no dataset *ICPR2012,* obtidos com a arquitetura *U-NET.*

Para o exemplo apresentado na Figura 53A, são apresentadas três informações, nomeadamente, a imagem original do dataset submetida a teste, a respetiva máscara onde se verificam as células que efetivamente se encontram em mitose, e a predição realizada com o modelo implementado. Analisando os resultados visuais para este exemplo (Figura 53A), observa-se que a célula em

mitose foi identificada com sucesso. Contudo, alguns falsos positivos foram detetados de modo menos nítido.

Relativamente ao segundo exemplo apresentado (Figura 53B), é também apresentada a imagem original do dataset submetida a teste, a predição realizada com o modelo implementado, e ainda, uma sobreposição da imagem testada com a predição, com a adição de uma caixa de deteção das áreas comum, ou seja, a mitose. Analisando os resultados visuais para este exemplo (Figura 53B), há novamente uma identificação correta da célula em mitose sem qualquer falso resultado. Visualmente, podemos concluir que os resultados são bastante promissores. Contudo, a validação dos mesmos através de métricas adequadas, nomeadamente o IoU e o coeficiente de DICE, são imprescindíveis para validar a metodologia utilizada.

4.3 Aplicação Web: Demonstração de resultados

Uma aplicação web foi desenvolvida com o intuito de tornar o processo de teste e visualização de resultados interativo. Na página inicial da aplicação (Figura 54) é possível aceder às páginas oficiais de cada um dos dataset que pode ser utilizado, para obter mais informações sobre os mesmos. A aplicação permite a escolha do dataset a testar (Figura 54 – menu lateral esquerdo), que depois de escolhido inicia o processo de aquisição de dados.

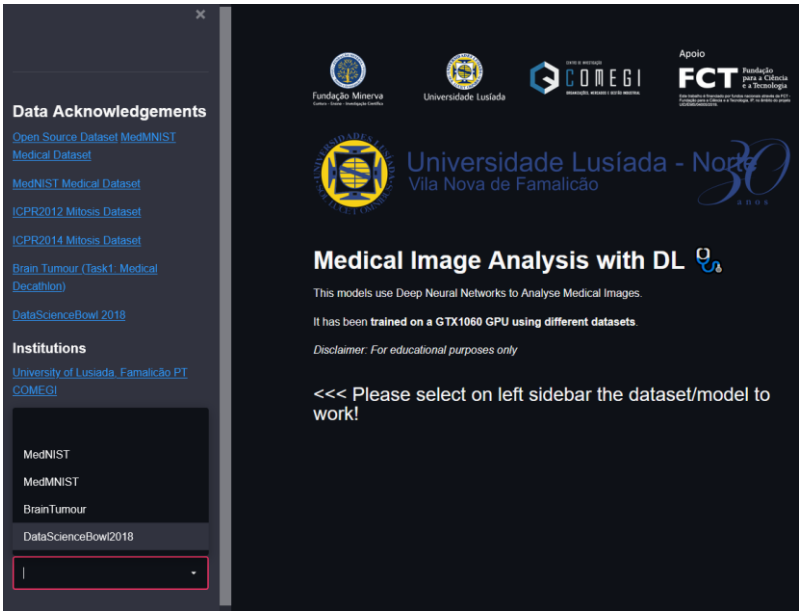


Figura 54. Página inicial da aplicação web.

Dependendo do dataset escolhido, o utilizador é questionado sobre o modelo que pretende usar para o processo de teste (Figura 55 – menu lateral esquerdo). A escolha do modelo a utilizar é limitada aos modelos utilizados com o respetivo conjunto de dados.

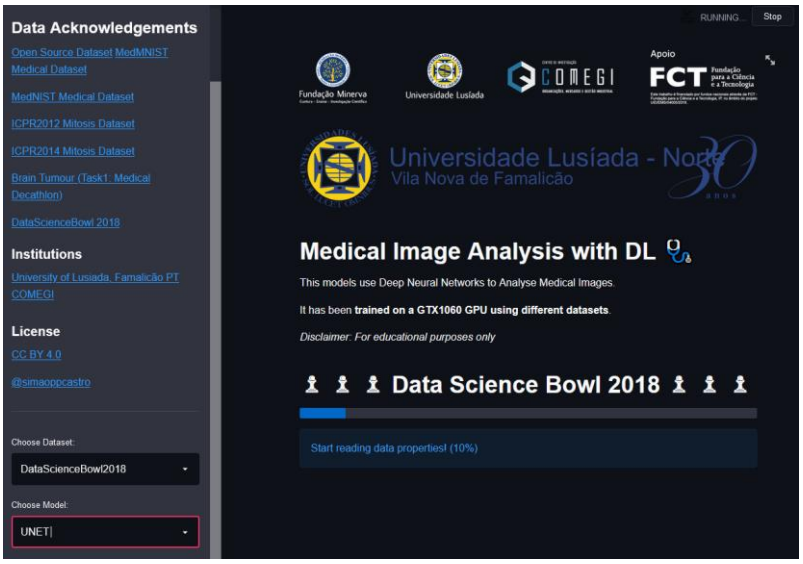


Figura 55. Processo de escolha do modelo para testar dataset escolhido.

Depois da aquisição dos dados e de carregado o modelo, é questionado ao utilizador em que parte do conjunto de dados este vai querer testar, assim como, o índice da imagem que quer testar (Figura 56).

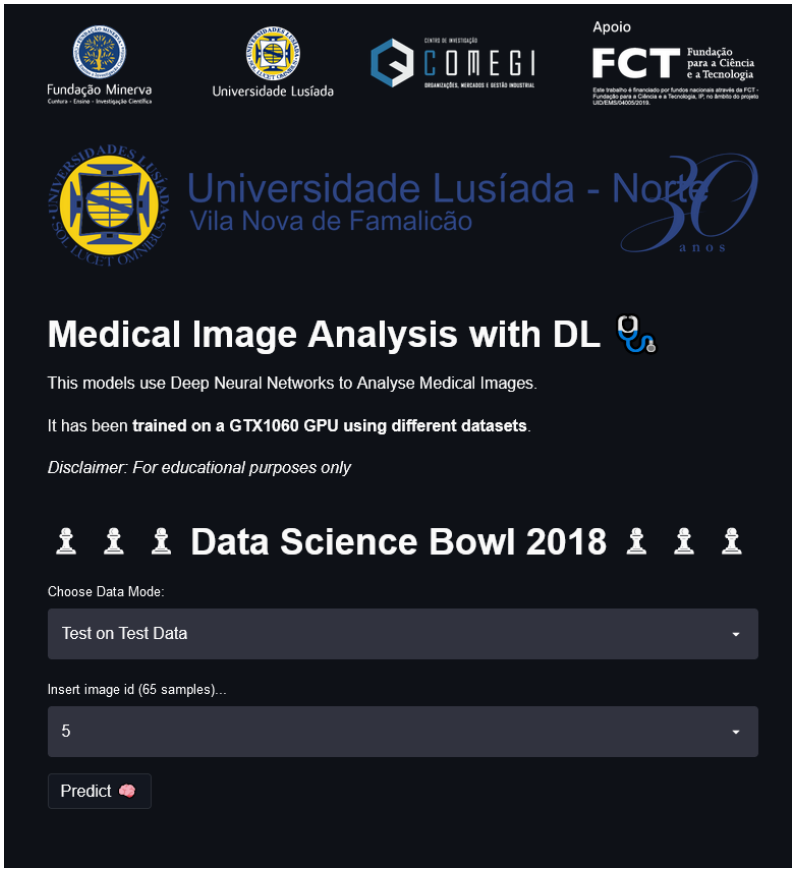


Figura 56. Processo de escolha do conjunto parcial de dados usado para testar na plataforma web.

Ao acionar a execução do teste (Figura 56 – botão *Predict*) é devolvido um resultado visual. Na Figura 57 é observado o resultado obtido no conjunto de dados *DSB2018*, onde foi usada uma imagem do conjunto de dados de teste, imagem número 5, através do modelo *UNET.*

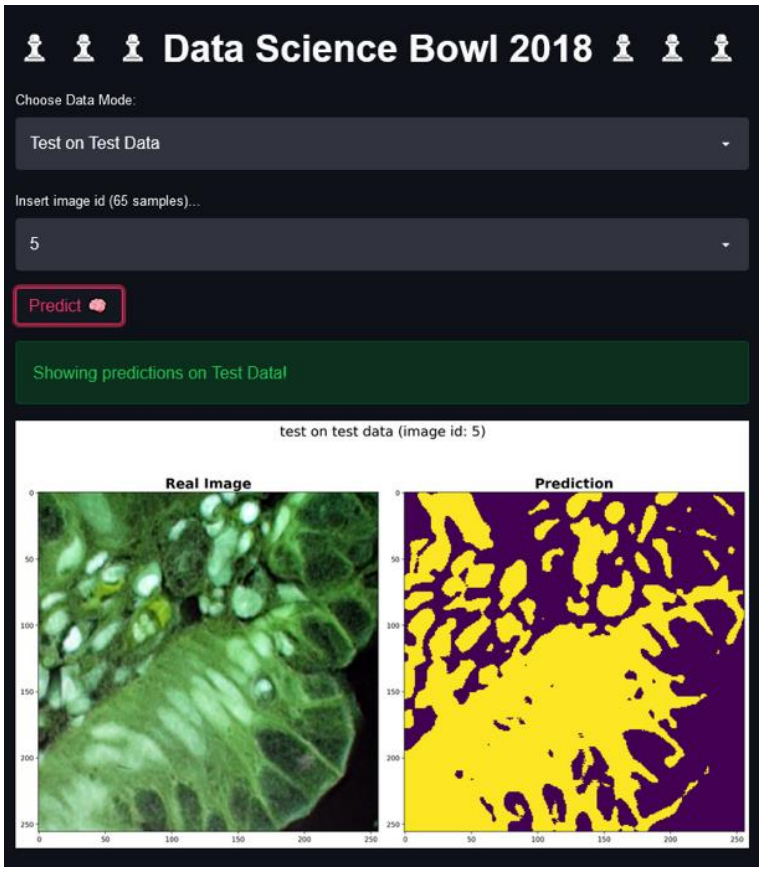


Figura 57. Exemplo de um resultado visual obtido na plataforma web com dados de teste do *DSB2018*.

Na Figura 58 é possível observar um outro exemplo, desta vez com o conjunto de dados usado na validação.

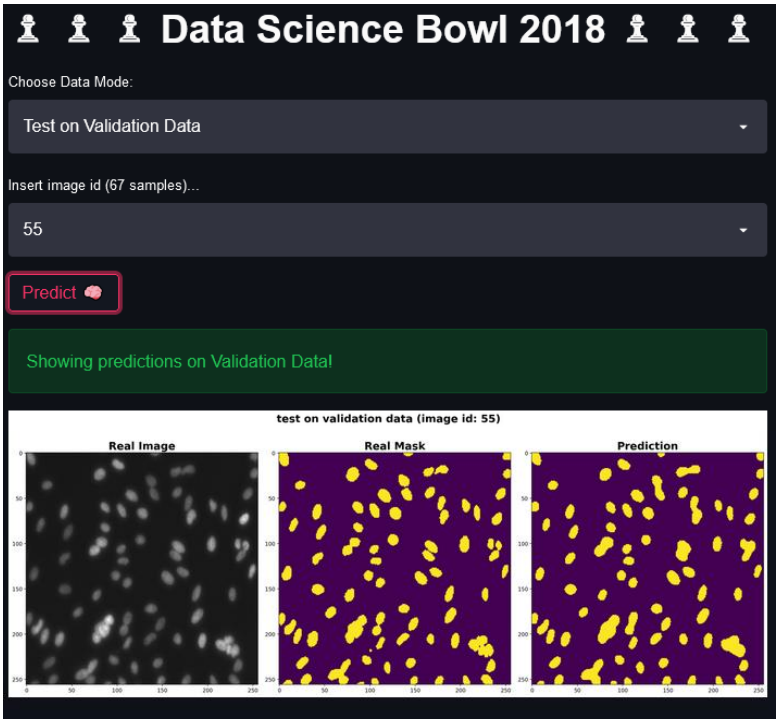


Figura 58. Exemplo de um resultado visual obtido na plataforma web com dados de validação do *DSB2018*.

Capítulo 5: Conclusões e Perspetivas Futuras

Nesta dissertação foram investigadas metodologias de *machine learning,* um ramo da inteligência artificial, para a análise de imagens médicas, nomeadamente, a classificação e a segmentação. Numa primeira abordagem, após selecionados os dataset, as imagens foram pré-processadas para que fossem corrigidas ou realçadas adequadamente, melhorando o seu contraste, corrigindo pixels defeituosos ou reduzindo ruído. Este tratamento das imagens é importante uma vez que a visualização direta de imagens médicas digitais geralmente não é adequada. Seguiu-se a investigação, implementação, teste e avaliação de desempenho dos modelos e parâmetros selecionados para a realização das duas tarefas propostas.

Tendo em consideração os resultados, pode-se concluir que as metodologias desenvolvidas, tanto para classificação como para segmentação demonstram grande potencial tendo sido obtidos resultados melhores quando comparados com os reportados na literatura. De facto, na classificação de imagens do dataset *MedMNIST* foram obtidos valores de exatidão dentro da mesma gama ou melhores que os valores previamente reportados por Yang *et al*. (2020) [237]. Para a segmentação de imagens com o dataset *DSB2018* é importante realçar que, os valores obtidos de IoU (0.9675 - 0.9843) são melhores que os reportados na literatura (0.9077 - 0.9263) [198], assim como os valores de coeficiente de DICE obtidos (0.9835 - 0.9916) relativamente ao valor reportado (0.9215) [240]. As configurações, transformações e o processamento de imagem aplicado aos dataset foram fator diferenciador nos resultados obtidos em ambas as tarefas.

As maiores dificuldades inerentes ao desenvolvimento deste trabalho estão associadas ao facto dos modelos utilizados nas diferentes tarefas requererem uma grande capacidade de processamento, o que limita a execução das tarefas em questões temporais de execução.

O trabalho desenvolvido e o estudo realizado sugerem diversas abordagens adicionais para o aprofundamento e melhoramento em termos de performance em trabalhos futuros. Como complemento ao trabalho desenvolvido seria importante aprofundar o trabalho desenvolvido para a identificação de estruturas específicas a nível celular, para incluir avaliação do desempenho desta tarefa através de métricas apropriadas. Adicionalmente, no contexto do trabalho de segmentação desenvolvido, a identificação das diferentes etapas da mitose (prófase, metáfase, anáfase, telófase) deverá contribuir cumulativamente para um melhor entendimento da aplicabilidade destas metodologias. Um outro caminho a explorar é a quantificação de núcleos presentes em amostras e desenvolver mecanismos de seleção automática de modelos e parâmetros de processamento a usar em dataset complexos.

Os objetivos definidos para este trabalho foram cumpridos e os resultados obtidos demonstram a importância das técnicas de inteligência artificial no apoio ao diagnóstico médico, permitindo que o processo de análise de imagem se torne uma tarefa autónoma e automática, permitindo à comunidade médica uma maior rapidez e eficiência no diagnóstico.

Referências

[1] J. Beutel, H. L. Kundel, and R. L. Van Metter, *Handbook of Medical Imaging*. Bellingham, Washington Spie Press, 2000.

[2] E. A. Krupinski, "The importance of perception research in medical imaging," *Radiat. Med.,* vol. 18, no. 6, pp. 329-334, 2000.

[3] H. Sung *et al.*, "Global cancer statistics 2020: GLOBOCAN estimates of incidence and mortality worldwide for 36 cancers in 185 countries," *CA: Cancer J. Clin.,* vol. 71, no. 3, pp. 209-249, Feb. 2021, doi: 10.3322/caac.21660.

[4] C. K. Kuhl, "The changing world of breast cancer: a radiologist's perspective," *Investig. Radiol.,* vol. 50, no. 9, pp. 615-628, Sep. 2015, doi: 10.1097/RLI.0000000000000166.

[5] J. Tum and A. Middleton, *Radiography of cultural material*. Routledge, 2006.

[6] H. Zaidi, *Quantitative analysis in nuclear medicine imaging*. Boston, MA: Springer, 2006.

[7] M. T. Vlaardingerbroek and J. A. Boer, *Magnetic resonance imaging: theory and practice*. Springer Science & Business Media, 2013.

[8] S. A. Morris and T. C. Slesnick, "Magnetic resonance imaging," *Visual Guide to Neonatal Cardiology,* pp. 104-108, Feb. 2018, doi: 10.1002/9781118635520.ch16.

[9] T. L. Szabo, *Diagnostic ultrasound imaging: inside out*. Burlington, MA, USA: Academic Press, 2004, p. 560.

[10] G. Berci and K. A. Forde, "History of endoscopy," *Surg. Endosc.,* vol. 14, no. 1, pp. 1-15, Jan. 2000, doi: 10.1007/s004649900002.

[11] T. A. Turner, "Diagnostic thermography," *Vet. Clin. North Am. Equine Pract.,* vol. 17, no. 1, pp. 95-114, Apr. 2001, doi: 10.1016/S0749-0739(17)30077-9.

[12] V. Gulshan *et al.*, "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs," *JAMA,* vol. 316, no. 22, pp. 2402-2410, Dec. 2016, doi: 10.1001/jama.2016.17216.

[13] EyePACS. http://www.eyepacs.com/. Acedido a 01.2021.

[14] ADCIS. https://www.adcis.net/en/third-party/messidor2/. Acedido a 01.2021.

[15] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Las Vegas, NV, USA, 2016, pp. 2818-2826, doi: 10.1109/CVPR.2016.308.

[16] L. P. Google AI Blog. https://ai.googleblog.com/2016/11/deep-learning-for-detection-of-diabetic.html. Acedido a 05.2021.

[17] T. Mathew, J. R. Kini, and J. Rajan, "Computational methods for automated mitosis detection in histopathology images: A review," *Biocybern. Biomed. Eng.,* vol. 41, no. 1, pp. 64-82, Mar. 2020, doi: 10.1016/j.bbe.2020.11.005.

[18] K. Sirinukunwattana, S. E. A. Raza, Y.-W. Tsang, D. R. Snead, I. A. Cree, and N. M. Rajpoot, "Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images," *IEEE Trans. Med. Imaging,* vol. 35, no. 5, pp. 1196-1206, Feb. 2016, doi: 10.1109/TMI.2016.2525803.

[19] J. A. Quinn, R. Nakasi, P. K. Mugagga, P. Byanyima, W. Lubega, and A. Andama, "Deep convolutional neural networks for microscopy-based point of care diagnostics," in *Machine Learning for Healthcare Conference*, 2016: PMLR, pp. 271-281.

[20] X. Jia and M. Q.-H. Meng, "A deep convolutional neural network for bleeding detection in wireless capsule endoscopy images," in *2016 38th annual international conference of the IEEE engineering in medicine and biology society (EMBC)*, Orlando, FL, USA, Aug. 2016: IEEE, pp. 639-642, doi: 10.1109/EMBC.2016.7590783.

[21] R. Leenhardt *et al.*, "A neural network algorithm for detection of GI angiectasia during small-bowel capsule endoscopy," *Gastrointest. Endosc.,* vol. 89, no. 1, pp. 189-194, Jun. 2019, doi: 10.1016/j.gie.2018.06.036.
[22] G. Litjens *et al.*, "State-of-the-art deep learning in cardiovascular image analysis," *JACC Cardiovasc. Imaging,* vol. 12, no. 8 Part 1, pp. 1549-1565, Aug. 2019, doi: 10.1016/j.jcmg.2019.06.009.
[23] H. F. Dvorak, "Tumors: wounds that do not heal," *N. Engl. J. Med.,* vol. 315, no. 26, pp. 1650-1659, Dec. 1986, doi: 10.1056/NEJM198612253152606.
[24] Z. Wang, G. Yu, Y. Kang, Y. Zhao, and Q. Qu, "Breast tumor detection in digital mammography based on extreme learning machine," *Neurocomputing,* vol. 128, pp. 175-184, Mar. 2014, doi: 10.1016/j.neucom.2013.05.053.
[25] H. Yan, "Remote sensing image classification based on SVM classifier," in *2011 International Conference on System science, Engineering design and Manufacturing informatization*, Guiyang, China, Oct. 2011, vol. 1: IEEE, pp. 30-33, doi: 10.1109/ICSSEM.2011.6081213.
[26] S. Sarraf and G. Tofighi, "Classification of alzheimer's disease using fmri data and deep learning convolutional neural networks," *arXiv preprint arXiv:1603.08631,* Mar. 2016.
[27] A. Ortiz, J. Munilla, M. Martínez-Ibañez, J. M. Górriz, J. Ramírez, and D. Salas-Gonzalez, "Parkinson's disease detection using isosurfaces-based features and convolutional neural networks," *Front. Neuroinform.,* vol. 13, p. 48, Jul. 2019, doi: 10.3389/fninf.2019.00048.
[28] P. Khatamino, İ. Cantürk, and L. Özyılmaz, "A Deep learning-CNN based system for medical diagnosis: an application on Parkinson's disease handwriting drawings," in *2018 6th International Conference on Control Engineering & Information Technology (CEIT)*, Istanbul, Turkey, Oct. 2018: IEEE, pp. 1-6, doi: 10.1109/CEIT.2018.8751879.
[29] M. Kantardzic, *Data mining: concepts, models, methods, and algorithms*. Hoboken, New Jersey: John Wiley & Sons, 2011.
[30] D. J. Hand, "Principles of data mining," *Drug Saf.,* vol. 30, no. 7, pp. 621-622, Jul. 2007, doi: 10.2165/00002018-200730070-00010.
[31] J. Jackson, "Data mining; a conceptual overview," *Commun. Assoc. Inf. Syst.,* vol. 8, no. 1, p. 19, 2002, doi: 10.17705/1CAIS.00819.
[32] S. Patro and K. K. Sahu, "Normalization: A preprocessing stage," *arXiv preprint arXiv:1503.06462,* Mar. 2015.
[33] S. A. Alasadi and W. S. Bhaya, "Review of data preprocessing techniques in data mining," *J. Eng. Appl. Sci.,* vol. 12, no. 16, pp. 4102-4107, Sept. 2017.
[34] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "Knowledge Discovery and Data Mining: Towards a Unifying Framework," in *KDD*, Aug. 1996, vol. 96, pp. 82-88.
[35] J. Han, M. Kamber, and J. Pei, "Data mining concepts and techniques third edition," in *The Morgan Kaufmann Series in Data Management Systems*, vol. 5, no. 4), 2011, pp. 83-124.
[36] S. Džeroski, "Multi-relational data mining: an introduction," *ACM SIGKDD Explorations Newsletter,* vol. 5, no. 1, pp. 1-16, Jul. 2003, doi: 10.1145/959242.959245.
[37] N. A. Fonseca, V. S. Costa, and R. Camacho, "Conceptual clustering of multi-relational data," in *International Conference on Inductive Logic Programming*, Windsor Great Park, United Kingdom, 2011: Springer, pp. 145-159, doi: 10.1007/978-3-642-31951-8_16.
[38] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI Mag.,* vol. 17, no. 3, pp. 37-37, Mar. 1996, doi: 10.1609/aimag.v17i3.1230.
[39] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science,* vol. 290, no. 5500, pp. 2323-2326, Dec. 2000, doi: 10.1126/science.290.5500.2323.

[40] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdiscip. Rev. Comput. Stat.,* vol. 2, no. 4, pp. 433-459, Jul. 2010, doi: 10.1002/wics.101.
[41] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Comput.,* vol. 10, no. 5, pp. 1299-1319, Jul. 1998, doi: 10.1162/089976698300017467.
[42] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.,* vol. 21, no. 9, pp. 1263-1284, Sept. 2009, doi: 10.1109/TKDE.2008.239.
[43] H. He and Y. Ma, *Imbalanced learning: foundations, algorithms, and applications*, 1 ed. 2013.
[44] N. Thai-Nghe, Z. Gantner, and L. Schmidt-Thieme, "Cost-sensitive learning methods for imbalanced data," in *The 2010 International joint conference on neural networks (IJCNN)*, Barcelona, Spain, Jul. 2010: IEEE, pp. 1-8, doi: 10.1109/IJCNN.2010.5596486.
[45] C. Sammut and G. I. Webb, *Encyclopedia of machine learning*. Sydney: Springer Science & Business Media, 2011.
[46] B. Krishnapuram, S. Yu, and R. B. Rao, *Cost-sensitive machine learning*. Broken Sound Parkway, NW: CRC Press, 2011.
[47] V. López, A. Fernández, S. García, V. Palade, and F. Herrera, "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics," *Inf. Sci.,* vol. 250, pp. 113-141, Nov. 2013, doi: 10.1016/j.ins.2013.07.007.
[48] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *J. Artif. Intell. Res.,* vol. 16, pp. 321-357, Jun. 2002, doi: 10.1613/jair.953.
[49] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, Hong Kong, China, Jun. 2008: IEEE, pp. 1322-1328, doi: 10.1109/IJCNN.2008.4633969.
[50] Y. E. Kurniawati, A. E. Permanasari, and S. Fauziati, "Adaptive synthetic-nominal (adasyn-n) and adaptive synthetic-knn (adasyn-knn) for multiclass imbalance learning on laboratory test data," in *2018 4th International Conference on Science and Technology (ICST)*, Yogyakarta, Indonesia, Aug. 2018: IEEE, pp. 1-6, doi: 10.1109/ICSTC.2018.8528679.
[51] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter,* vol. 6, no. 1, pp. 20-29, Jun. 2004, doi: 10.1145/1007730.1007735.
[52] B. V. Reddy, P. B. Reddy, P. S. Kumar, and S. S. Reddy, "Developing an approach to brain MRI image preprocessing for tumor detection," *Int. J. Res,* vol. 1, no. 6, pp. 725-731, Jul. 2014.
[53] M. B. A. Miah and M. A. Yousuf, "Detection of lung cancer from CT image using image processing and neural network," in *2015 International conference on electrical engineering and information communication technology (ICEEICT)*, Savar, Bangladesh, May. 2015: IEEE, pp. 1-6, doi: 10.1109/ICEEICT.2015.7307530.
[54] D. N. Ponraj, M. E. Jenifer, P. Poongodi, and J. S. Manoharan, "A survey on the preprocessing techniques of mammogram for the detection of breast cancer," *J. Emerg. Trends Comput. Inf. Sci.,* vol. 2, no. 12, pp. 656-664, Dec. 2011.
[55] Y. Zhang, R. Sankar, and W. Qian, "Boundary delineation in transrectal ultrasound image for prostate cancer," *Comput. Biol. Med.,* vol. 37, no. 11, pp. 1591-1599, Nov. 2007, doi: 10.1016/j.compbiomed.2007.02.008.
[56] J. Sikorski, "Identification of malignant melanoma by wavelet analysis," *Proceedings of Student/Faculty Research Day, CSIS, Pace University,* May. 2004.
[57] A. Chiem, A. Al-Jumaily, and R. N. Khushaba, "A novel hybrid system for skin lesion detection," in *2007 3rd International Conference on Intelligent Sensors, Sensor Networks and Information*, Dec. 2007: IEEE, pp. 567-572, doi: 10.1109/ISSNIP.2007.4496905.

[58] I. Maglogiannis, E. Zafiropoulos, and C. Kyranoudis, "Intelligent segmentation and classification of pigmented skin lesions in dermatological images," in *Hellenic Conference on Artificial Intelligence*, 2006, vol. 3955: Springer, pp. 214-223, doi: 10.1007/11752912_23.

[59] T. Tanaka, S. Torii, I. Kabuta, K. Shimizu, and M. Tanaka, "Pattern classification of nevus with texture analysis," *IEEJ Trans. Electr. Electron. Eng.,* vol. 3, no. 1, pp. 143-150, Dec. 2008, doi: 10.1002/tee.20246.

[60] H. Zhou, M. Chen, and J. M. Rehg, "Dermoscopic interest point detector and descriptor," in *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, Boston, MA, USA, Jun. 2009: IEEE, pp. 1318-1321, doi: 10.1109/ISBI.2009.5193307.

[61] C. Lee and D. A. Landgrebe, "Feature extraction based on decision boundaries," *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 15, no. 4, pp. 388-400, Apr. 1993, doi: 10.1109/34.206958.

[62] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN Computer Science,* vol. 2, no. 3, pp. 1-21, Mar. 2021, doi: 10.1007/s42979-021-00592-x.

[63] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information processing & management,* vol. 45, no. 4, pp. 427-437, Jul. 2009, doi: 10.1016/j.ipm.2009.03.002.

[64] Y. Jiao and P. Du, "Performance measures in evaluating machine learning based bioinformatics predictors for classifications," *Quant. Biol.,* vol. 4, no. 4, pp. 320-330, Oct. 2016, doi: 10.1007/s40484-016-0081-2.

[65] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Long Beach, CA, USA, Jun. 2019, pp. 658-666, doi: 10.1109/CVPR.2019.00075.

[66] J. Bertels *et al.*, "Optimizing the dice score and jaccard index for medical image segmentation: Theory and practice," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Oct. 2019, vol. 11765: Springer, pp. 92-100, doi: 10.1007/978-3-030-32245-8_11.

[67] S. Jadon, "A survey of loss functions for semantic segmentation," in *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, Via del Mar, Chile, Oct. 2020: IEEE, pp. 1-7, doi: 10.1109/CIBCB48159.2020.9277638.

[68] M. Yi-de, L. Qing, and Q. Zhi-Bai, "Automated image segmentation using improved PCNN model based on cross-entropy," in *Proceedings of 2004 International Symposium on Intelligent Multimedia, Video and Speech Processing, 2004.*, Hong Kong, China, Oct. 2004: IEEE, pp. 743-746, doi: 10.1109/ISIMP.2004.1434171.

[69] P. Christoffersen and K. Jacobs, "The importance of the loss function in option valuation," *J. Financ. Econ.,* vol. 72, no. 2, pp. 291-318, May. 2004, doi: 10.1016/j.jfineco.2003.02.001.

[70] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. J. Cardoso, "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations," in *Deep learning in medical image analysis and multimodal learning for clinical decision support*, vol. 10553: Springer, 2017, pp. 240-248.

[71] R. K. Amin and Y. Sibaroni, "Implementation of decision tree using C4. 5 algorithm in decision making of loan application by debtor (Case study: Bank pasar of Yogyakarta Special Region)," in *2015 3rd International Conference on Information and Communication Technology (ICoICT)*, Nusa Dua, Bali, Indonesia, May. 2015: IEEE, pp. 75-80, doi: 10.1109/ICoICT.2015.7231400.

[72] J. Gama and L.-I. Porto, "Bayesian learning: An introduction.," *University of Porto: Porto, Portugal,* 2008.

[73] K. Kourou, T. P. Exarchos, K. P. Exarchos, M. V. Karamouzis, and D. I. Fotiadis, "Machine learning applications in cancer prognosis and prediction," *Comput. Struct. Biotechnol. J.,* vol. 13, pp. 8-17, Nov. 2015, doi: 10.1016/j.csbj.2014.11.005.

[74] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering,* vol. 160, no. 1, pp. 3-24, 2007.

[75] I. H. Witten and E. Frank, "Data mining: practical machine learning tools and techniques with Java implementations," *ACM SIGMOD Record,* vol. 31, no. 1, pp. 76-77, Mar. 2002.

[76] H. Drucker, D. Wu, and V. N. Vapnik, "Support vector machines for spam categorization," *IEEE Trans. Neural Netw. Learn. Syst.,* vol. 10, no. 5, pp. 1048-1054, Sept. 1999, doi: 10.1109/72.788645.

[77] M. Vatsa, R. Singh, and A. Noore, "Improving biometric recognition accuracy and robustness using DWT and SVM watermarking," *IEICE Electron. Express,* vol. 2, no. 12, pp. 362-367, 2005, doi: 10.1587/elex.2.362.

[78] E. Byvatov and G. Schneider, "Support vector machine applications in bioinformatics," *Appl. Bioinformatics,* vol. 2, no. 2, pp. 67-77, Jan. 2003.

[79] A. Tharwat, A. E. Hassanien, and B. E. Elnaghi, "A BA-based algorithm for parameter optimization of support vector machine," *Pattern. Recognit. Lett.,* vol. 93, pp. 13-22, Jul. 2017, doi: 10.1016/j.patrec.2016.10.007.

[80] R. Zhang and J. Ma, "An improved SVM method P-SVM for classification of remotely sensed data," *Int. J. Remote Sens.,* vol. 29, no. 20, pp. 6029-6036, Sept. 2008, doi: 10.1080/01431160802220151.

[81] R. Eberhart and R. Dobbins, "Early neural network development history: The age of Camelot," *IEEE Eng. Med. Biol. Mag.,* vol. 9, no. 3, pp. 15-18, Sep. 1990, doi: 10.1109/51.59207.

[82] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.,* vol. 5, no. 4, pp. 115-133, Dec. 1943, doi: 10.1007/BF02478259.

[83] D. O. Hebb, "The organization of behavior; a neuropsycholocigal theory," *A Wiley Book in Clinical Psychology,* vol. 62, p. 78, 1949, doi: 10.2307/1418888.

[84] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological review,* vol. 65, no. 6, p. 386, 1958.

[85] B. Widrow and M. E. Hoff, "Associative storage and retrieval of digital information in networks of adaptive “neurons”," in *Biological Prototypes and Synthetic Systems*. Boston, MA: Springer, 1962, pp. 160-160.

[86] M. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[87] J. Werbose, "Beyond regression: new tools for prediction and analysis in the behavioral," 1974.

[88] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE Trans. Syst. Man Cybern.,* no. 5, pp. 826-834, Sept. 1983, doi: 10.1109/TSMC.1983.6313076.

[89] M. I. Jordan, "Serial order: A parallel distributed processing approach," in *Advances in psychology*, vol. 121: Elsevier, 1997, pp. 471-495.

[90] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.,* vol. 1, no. 4, pp. 541-551, Dec. 1989, doi: 10.1162/neco.1989.1.4.541.

[91] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Adv. Neural Inf. Process. Syst.,* vol. 19, pp. 153-160, 2007.

[92] B. Macukow, "Neural networks–state of art, brief history, basic models and architecture," in *IFIP international conference on computer information systems and industrial management*, Sept. 2016: Springer, pp. 3-14, doi: 10.1007/978-3-319-45378-1_1.

[93] K. J. Cios, "Deep neural networks—a brief history," in *Advances in Data Analysis with Computational Intelligence Methods*, vol. 738: Springer, 2018, pp. 183-200.
[94] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning* (no. 2). Cambridge, MA, USA: MIT press Cambridge, 2016.
[95] B. Shickel, P. J. Tighe, A. Bihorac, and P. Rashidi, "Deep EHR: a survey of recent advances in deep learning techniques for electronic health record (EHR) analysis," *IEEE J. Biomed. Health Inform.,* vol. 22, no. 5, pp. 1589-1604, Sep. 2017, doi: 10.1109/JBHI.2017.2767063.
[96] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon,* vol. 4, no. 11, p. e00938, Nov. 2018, doi: 10.1016/j.heliyon.2018.e00938.
[97] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 31, no. 5, pp. 855-868, May. 2008, doi: 10.1109/TPAMI.2008.137.
[98] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychol. Rev.,* vol. 65, no. 6, pp. 386-408, 1958, doi: 10.1037/h0042519.
[99] D. S. Chen and R. C. Jain, "A robust backpropagation learning algorithm for function approximation," *IEEE Trans. Neural Netw. Learn. Syst.,* vol. 5, no. 3, pp. 467-479, May 1994, doi: 10.1109/72.286917.
[100] J. Li, J.-h. Cheng, J.-y. Shi, and F. Huang, "Brief introduction of back propagation (BP) neural network algorithm and its improvement," in *Advances in computer science and information engineering*, vol. 169. Berlin, Heidelberg: Springer, 2012, pp. 553-558.
[101] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747,* Jun. 2016.
[102] I. Goodfellow and Y. Bengio, "Courville. A (2016) Deep learning," *MIT Press.(online available),* Oct. 2016, doi: 10.4258/hir.2016.22.4.351.
[103] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378,* p. 20, Nov. 2018.
[104] J. Cao, K. Zhang, H. Yong, X. Lai, B. Chen, and Z. Lin, "Extreme learning machine with affine transformation inputs in an activation function," *EEE Trans. Neural Netw. Learn. Syst.,* vol. 30, no. 7, pp. 2093-2107, Nov. 2018, doi: 10.1109/TNNLS.2018.2877468.
[105] C. Olah, A. Mordvintsev, and L. Schubert, "Feature visualization," *Distill,* vol. 2, no. 11, p. e7, Nov. 2017, doi: 10.23915/distill.00007.
[106] W. A. Little, "The existence of persistent states in the brain," *Math. Biosci.,* vol. 19, no. 1-2, pp. 101-120, Feb. 1974, doi: 10.1016/0025-5564(74)90031-5.
[107] J. Turian, J. Bergstra, and Y. Bengio, "Quadratic features and deep architectures for chunking," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, Jun. 2009, pp. 245-248.
[108] W. A. Little and G. L. Shaw, "Analytic study of the memory storage capacity of a neural network," *Math. Biosci.,* vol. 39, no. 3-4, pp. 281-290, Jun. 1978, doi: 10.1016/0025-5564(78)90058-5.
[109] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature,* vol. 521, no. 7553, pp. 436-444, 2015.
[110] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.
[111] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, "Incorporating second-order functional knowledge for better option pricing," *Adv. Neural Inf. Process. Syst.,* vol. 12, pp. 472-478, 2001.

[112] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289,* Nov. 2015.
[113] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *International conference on machine learning*, May. 2013: PMLR, pp. 1319-1327.
[114] M. Basirat and P. M. Roth, "The quest for the golden activation function," *arXiv preprint arXiv:1808.00783,* Aug. 2018.
[115] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941,* Oct. 2017.
[116] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *International Workshop on Artificial Neural Networks*, Jun. 1995, vol. 930: Springer, pp. 195-201, doi: 10.1007/3-540-59497-3_175.
[117] R. M. Neal, "Connectionist learning of belief networks," *Artif. Intell.,* vol. 56, no. 1, pp. 71-113, Jul. 1992, doi: 10.1016/0004-3702(92)90065-6.
[118] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized MLP architectures of neural networks," *Int. J. Artif. Intell. Expert Syst.,* vol. 1, no. 4, pp. 111-122, Feb. 2011.
[119] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Adv. Neural Inf. Process. Syst.,* vol. 25, pp. 1097-1105, 2012.
[120] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 39, no. 12, pp. 2481-2495, Jan. 2017, doi: 10.1109/TPAMI.2016.2644615.
[121] P. Le and W. Zuidema, "Compositional distributional semantics with long short term memory," *arXiv preprint arXiv:1503.02510,* Mar. 2015.
[122] W. Ping *et al.*, "Deep voice 3: Scaling text-to-speech with convolutional sequence learning," *arXiv preprint arXiv:1710.07654,* Nov. 2017.
[123] M. D. Zeiler *et al.*, "On rectified linear units for speech processing," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, BC, Canada, May. 2013: IEEE, pp. 3517-3521, doi: 10.1109/ICASSP.2013.6638312.
[124] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in *2013 IEEE international conference on acoustics, speech and signal processing*, Vancouver, BC, Canada, May. 2013: IEEE, pp. 8609-8613, doi: 10.1109/ICASSP.2013.6639346.
[125] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, "Incorporating second-order functional knowledge for better option pricing," *Adv. Neural Inf. Process. Syst.,* pp. 472-478, 2001.
[126] L. Trottier, P. Giguere, and B. Chaib-Draa, "Parametric exponential linear unit for deep convolutional neural networks," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Cancun, Mexico, Dec. 2017: IEEE, pp. 207-214, doi: 10.1109/ICMLA.2017.00038.
[127] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *arXiv preprint arXiv:1706.02515,* Sept. 2017.
[128] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Miami, FL, USA, Aug. 2009: IEEE, pp. 248-255, doi: 10.1109/CVPR.2009.5206848.
[129] J. Fu, H. Zheng, and T. Mei, "Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Honolulu, HI, USA, Jul. 2017, pp. 4438-4446, doi: 10.1109/CVPR.2017.476.

[130] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861,* Apr. 2017.
[131] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Honolulu, Hawaii, 2017, pp. 1492-1500.
[132] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770-778.
[133] V. Golkov *et al.*, "Q-space deep learning: twelve-fold shorter and model-free diffusion MRI scans," *IEEE Trans. Med. Imaging,* vol. 35, no. 5, pp. 1344-1351, Apr. 2016, doi: 10.1109/TMI.2016.2551324.
[134] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, Jun. 2015, vol. 37: PMLR, pp. 448-456.
[135] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556,* Sept. 2014.
[136] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, 2014, vol. 8689: Springer, pp. 818-833, doi: 10.1007/978-3-319-10590-1_53.
[137] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400,* Dec. 2013.
[138] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580,* Jul. 2012.
[139] P. Danaee, R. Ghaeini, and D. A. Hendrix, "A deep learning approach for cancer detection and relevant gene identification," in *Pacific symposium on biocomputing 2017*, 2017: World Scientific, pp. 219-229, doi: 10.1142/9789813207813_0022.
[140] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature,* vol. 323, no. 6088, pp. 533-536, Oct. 1986, doi: 10.1038/323533a0.
[141] S. Zhang, S. Zhang, B. Wang, and T. G. Habetler, "Deep learning algorithms for bearing fault Diagnostics —A comprehensive review," *IEEE Access,* vol. 8, pp. 29857-29881, Feb. 2020, doi: 10.1109/ACCESS.2020.2972859.
[142] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.,* vol. 11, no. 12, pp. 3371-3408, Oct. 2010.
[143] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114,* Dec. 2013.
[144] J. Zabalza *et al.*, "Novel segmented stacked autoencoder for effective dimensionality reduction and feature extraction in hyperspectral imaging," *Neurocomputing,* vol. 185, pp. 1-10, Apr. 2016, doi: 10.1016/j.neucom.2015.11.044.
[145] G. Hinton, "Boltzmann Machines," in *Encyclopedia of Machine Learning and Data Mining*, C. Sammut and G. I. Webb Eds. Boston, MA: Springer US, 2014, pp. 1-7.
[146] G. E. Hinton, "A practical guide to training restricted Boltzmann machines," in *Neural networks: Tricks of the trade*, vol. 7700: Springer, 2012, pp. 599-619.
[147] K. Swersky, B. Chen, B. Marlin, and N. De Freitas, "A tutorial on stochastic approximation algorithms for training restricted Boltzmann machines and deep belief nets," in *2010 Information Theory and Applications Workshop (ITA)*, La Jolla, CA, USA, Apr. 2010: IEEE, pp. 1-10, doi: 10.1109/ITA.2010.5454138.
[148] A. Fischer and C. Igel, "Training restricted Boltzmann machines: An introduction," *Pattern Recognit.,* vol. 47, no. 1, pp. 25-39, Jan. 2014, doi: 10.1016/j.patcog.2013.05.025.

[149] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.,* vol. 18, no. 7, pp. 1527-1554, Jul. 2006, doi: 10.1162/neco.2006.18.7.1527.
[150] K. Al-Jabery, T. Obafemi-Ajayi, G. Olbricht, and D. Wunsch, *Computational Learning Approaches to Data Analytics in Biomedical Applications*. Academic Press, 2019, p. 310.
[151] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks,* vol. 3361, no. 10, p. 1995, 1995.
[152] Y. B. Ian Goodfellow, Aaron Courville "Deep Learning," *The reference book for deep learning models,* 2016, doi: 10.4258/hir.2016.22.4.351.
[153] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *J. Physiol.,* vol. 160, no. 1, pp. 106-154, Jan. 1962, doi: 10.1113/jphysiol.1962.sp006837.
[154] T. Poggio and T. Serre, "Models of visual cortex," *Scholarpedia,* vol. 8, no. 4, p. 3516, 2013.
[155] V. Maeda-Gutierrez *et al.*, "Comparison of convolutional neural network architectures for classification of tomato plant diseases," *Appl. Sci.,* vol. 10, no. 4, p. 1245, Feb. 2020, doi: 10.3390/app10041245.
[156] S. Albelwi and A. Mahmood, "A framework for designing the architectures of deep convolutional neural networks," *Entropy,* vol. 19, no. 6, p. 242, May 2017, doi: 10.3390/e19060242.
[157] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International conference on artificial neural networks*, 2010, vol. 6354: Springer, pp. 92-101, doi: 10.1007/978-3-642-15825-4_10.
[158] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 37, no. 9, pp. 1904-1916, Jan. 2015, doi: 10.1109/TPAMI.2015.2389824.
[159] H. Wu and X. Gu, "Max-pooling dropout for regularization of convolutional neural networks," in *International Conference on Neural Information Processing*, Nov. 2015: Springer, pp. 46-54, doi: 10.1007/978-3-319-26532-2_6.
[160] W. Ouyang *et al.*, "Deepid-net: Deformable deep convolutional neural networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Boston, MA, USA, Jun. 2015, pp. 2403-2412, doi: 10.1109/CVPR.2015.7298854.
[161] Q. Zhao, S. Lyu, B. Zhang, and W. Feng, "Multiactivation pooling method in convolutional neural networks for image recognition," *Wirel. Commun. Mob. Comput.,* vol. 2018, Jun. 2018, doi: 10.1155/2018/8196906.
[162] F. Saeedan, N. Weber, M. Goesele, and S. Roth, "Detail-preserving pooling in deep networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, Jun. 2018: IEEE, pp. 9108-9116, doi: 10.1109/CVPR.2018.00949.
[163] K. Bhattacharjee, M. Pant, Y.-D. Zhang, and S. C. Satapathy, "Multiple Instance Learning with Genetic Pooling for medical data analysis," *Pattern. Recognit. Lett.,* vol. 133, pp. 247-255, May 2020, doi: 10.1016/j.patrec.2020.02.025.
[164] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Netw.,* vol. 6, no. 6, pp. 861-867, Oct. 1993, doi: 10.1016/S0893-6080(05)80131-5.
[165] S. Sonoda and N. Murata, "Neural network with unbounded activation functions is universal approximator," *Appl. Comput. Harmon. Anal.,* vol. 43, no. 2, pp. 233-268, Dec. 2017, doi: 10.1016/j.acha.2015.12.005.
[166] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.,* vol. 15, no. 1, pp. 1929-1958, Jun. 2014.

[167] R. K. Vinayak and R. Gilad-Bachrach, "Dart: Dropouts meet multiple additive regression trees," in *Artificial Intelligence and Statistics*, San Diego, CA, USA, 2015, vol. 38: PMLR, pp. 489-497.
[168] E. H. Houssein, M. M. Emam, A. A. Ali, and P. N. Suganthan, "Deep and machine learning techniques for medical imaging-based breast cancer: A comprehensive review," *Expert Syst. Appl.,* vol. 167, p. 114161, Apr. 2020, doi: 10.1016/j.eswa.2020.114161.
[169] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE,* vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.
[170] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Boston, MA, USA, Jun. 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.
[171] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, Feb. 2017, vol. 31, no. 1.
[172] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Honolulu, HI, USA, Jul. 2017: IEEE, pp. 1251-1258, doi: 10.1109/CVPR.2017.195.
[173] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artif. Intell. Rev.,* vol. 53, no. 8, pp. 5455-5516, Apr. 2020, doi: 10.1007/s10462-020-09825-6.
[174] Y. LeCun *et al.*, "Handwritten digit recognition with a back-propagation network," in *Advances in neural information processing systems*, 1990, pp. 396-404.
[175] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Honolulu, HI, USA, Jul. 2017, pp. 4700-4708, doi: 10.1109/CVPR.2017.243.
[176] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *arXiv preprint arXiv:1710.09829,* Nov. 2017.
[177] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications,* vol. 5, 2001.
[178] H. Wang and B. Raj, "On the origin of deep learning," *arXiv preprint arXiv:1702.07800,* Mar. 2017.
[179] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent," in *Backpropagation: Theory, architectures, and applications*, vol. 433, 1995, p. 17.
[180] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proc. IEEE,* vol. 78, no. 10, pp. 1550-1560, Oct. 1990, doi: 10.1109/5.58337.
[181] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw. Learn. Syst.,* vol. 5, no. 2, pp. 157-166, Mar. 1994, doi: 10.1109/72.279181.
[182] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, 2013: PMLR, pp. 1310-1318.
[183] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.,* vol. 9, no. 8, pp. 1735-1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.
[184] I. J. Goodfellow *et al.*, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661,* Jun. 2014.
[185] S. Kazeminia *et al.*, "GANs for medical image analysis," *Artif. Intell. Med.,* vol. 109, p. 101938, Sept. 2020, doi: 10.1016/j.artmed.2020.101938.
[186] D. Nie *et al.*, "Medical image synthesis with deep convolutional adversarial networks," *IEEE Trans. Biomed. Eng.,* vol. 65, no. 12, pp. 2720-2730, Mar. 2018, doi: 10.1109/TBME.2018.2814538.

[187] X. Yi, E. Walia, and P. Babyn, "Generative adversarial network in medical imaging: A review," *Med. Image. Anal.,* vol. 58, p. 101552, Dec. 2019, doi: 10.1016/j.media.2019.101552.
[188] X. Xu, S. Xu, L. Jin, and E. Song, "Characteristic analysis of Otsu threshold and its applications," *Pattern. Recognit. Lett.,* vol. 32, no. 7, pp. 956-961, May 2011, doi: 10.1016/j.patrec.2011.01.021.
[189] J. Fan, G. Zeng, M. Body, and M.-S. Hacid, "Seeded region growing: an extensive and comparative study," *Pattern. Recognit. Lett.,* vol. 26, no. 8, pp. 1139-1156, Jun. 2005, doi: 10.1016/j.patrec.2004.10.010.
[190] P. D. R. Raju and G. Neelima, "Image segmentation by using histogram thresholding," *Int. J. Eng. Comput. Sci.,* vol. 2, no. 1, pp. 776-779, Jan. 2012.
[191] M. Emre Celebi *et al.*, "Border detection in dermoscopy images using statistical region merging," *Skin. Res. Technol.,* vol. 14, no. 3, pp. 347-353, Jul. 2008, doi: 10.1111/j.1600-0846.2008.00301.x.
[192] N. Tong, H. Lu, X. Ruan, and M.-H. Yang, "Salient object detection via bootstrap learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Boston, MA, USA, Jun. 2015: IEEE, pp. 1884-1892, doi: 10.1109/CVPR.2015.7298798.
[193] B. Bozorgtabar, M. Abedini, and R. Garnavi, "Sparse coding based skin lesion segmentation using dynamic rule-based refinement," in *international workshop on machine learning in medical imaging*, Oct. 2016, vol. 10019: Springer, pp. 254-261, doi: 10.1007/978-3-319-47157-0_31.
[194] X. Li, Y. Li, C. Shen, A. Dick, and A. Van Den Hengel, "Contextual hypergraph modeling for salient object detection," in *Proceedings of the IEEE international conference on computer vision*, Dec. 2013, pp. 3328-3335.
[195] G. I. Sayed, M. A. Ali, T. Gaber, A. E. Hassanien, and V. Snasel, "A hybrid segmentation approach based on neutrosophic sets and modified watershed: a case of abdominal CT Liver parenchyma," in *2015 11th international computer engineering conference (ICENCO)*, Cairo, Egypt, Dec. 2015: IEEE, pp. 144-149, doi: 10.1109/ICENCO.2015.7416339.
[196] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, Nov. 2015, vol. 9351: Springer, pp. 234-241, doi: 10.1007/978-3-319-24574-4_28.
[197] T. Falk *et al.*, "U-Net: deep learning for cell counting, detection, and morphometry," *Nat. Methods,* vol. 16, no. 1, pp. 67-70, Dec. 2019, doi: 10.1038/s41592-018-0261-2.
[198] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang, "Unet++: A nested u-net architecture for medical image segmentation," in *Deep learning in medical image analysis and multimodal learning for clinical decision support*, vol. 11045: Springer, 2018, pp. 3-11.
[199] Z. Zhang, Q. Liu, and Y. Wang, "Road extraction by deep residual u-net," *IEEE Geosci. Remote S.,* vol. 15, no. 5, pp. 749-753, Mar. 2018, doi: 10.1109/LGRS.2018.2802944.
[200] Ö. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3D U-Net: learning dense volumetric segmentation from sparse annotation," in *International conference on medical image computing and computer-assisted intervention*, Oct. 2016, vol. 9901: Springer, pp. 424-432, doi: 10.1007/978-3-319-46723-8_49.
[201] Python. https://www.python.org/. Acedido a 01.2021.
[202] Tensorflow. https://www.tensorflow.org/. Acedido a 01.2021.
[203] PyTorch. https://pytorch.org/. Acedido a 01.2021.
[204] Numpy. https://numpy.org/. Acedido a 01.2021.
[205] pandas. https://pandas.pydata.org/. Acedido a 01.2021.
[206] Seaborn. https://seaborn.pydata.org/. Acedido a 01.2021.
[207] MatPlotlib. https://matplotlib.org/. Acedido a 01.2021.

[208] Scikit-learn. https://scikit-learn.org/. Acedido a 01.2021.
[209] Streamlit. https://docs.streamlit.io/en/stable/. Acedido a 04.2021.
[210] JetBrains. https://www.jetbrains.com/. Acedido a 01.2021.
[211] JetBrains. https://www.jetbrains.com/pycharm/. Acedido a 01.2021.
[212] J. Yang, R. Shi, and B. Ni, "MedMNIST Classification Decathlon: A Lightweight AutoML Benchmark for Medical Image Analysis," in *IEEE*, Nice, France, Apr. 2021, doi: 10.1109/ISBI48211.2021.9434062.
[213] C. E. Kahn Jr, "We All Need a Little Magic," ed: Radiological Society of North America, 2019.
[214] J. N. Kather *et al.*, "Predicting survival from colorectal cancer histology slides using deep learning: A retrospective multicenter study," *PLoS Med.,* vol. 16, no. 1, p. e1002730, Jan. 2019, doi: 10.1371/journal.pmed.1002730.
[215] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers, "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Jul. 2017, pp. 2097-2106.
[216] P. Tschandl, C. Rosendahl, and H. Kittler, "The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions," *Scientific data,* vol. 5, no. 1, pp. 1-9, Aug. 2018, doi: 10.1038/sdata.2018.161.
[217] D. S. Kermany *et al.*, "Identifying medical diagnoses and treatable diseases by image-based deep learning," *Cell,* vol. 172, no. 5, pp. 1122-1131. e9, Feb. 2018, doi: 10.1016/j.cell.2018.02.010.
[218] D. Dataset, "The 2nd diabetic retinopathy–grading and image quality estimation challenge," ed, 2020.
[219] W. Al-Dhabyani, M. Gomaa, H. Khaled, and A. Fahmy, "Dataset of breast ultrasound images," *Data in brief,* vol. 28, p. 104863, Feb. 2020, doi: 10.1016/j.dib.2019.104863.
[220] X. Xu, F. Zhou, B. Liu, D. Fu, and X. Bai, "Efficient multiple organ localization in ct image using 3d region proposal network," *IEEE Trans. Med. Imaging,* vol. 38, no. 8, pp. 1885-1898, Jan. 2019, doi: 10.1109/TMI.2019.2894854.
[221] K. Clark *et al.*, "The Cancer Imaging Archive (TCIA): maintaining and operating a public information repository," *J. Digit. Imaging,* vol. 26, no. 6, pp. 1045-1057, Dec. 2013, doi: 10.1007/s10278-013-9622-7.
[222] S. S. Halabi *et al.*, "The RSNA pediatric bone age machine learning challenge," *Radiology,* vol. 290, no. 2, pp. 498-503, Nov. 2019, doi: 10.1148/radiol.2018180736.
[223] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, 2013, vol. 28, no. 3: PMLR, pp. 1139-1147.
[224] PyTorch. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. Acedido a 01.2021.
[225] D. Verma, M. Kumar, and S. Eregala, "Deep Demosaicing using ResNet-Bottleneck Architecture," in *International Conference on Computer Vision and Image Processing*, Mar. 2019, vol. 1148: Springer, pp. 170-179, doi: 10.1007/978-981-15-4018-9_16.
[226] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* 2014.
[227] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101,* Jan. 2017.
[228] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, May. 2019: PMLR, pp. 6105-6114.

[229] Tensorflow. https://www.tensorflow.org/api_docs/python/tf/keras/applications/EfficientNetB0. Acedido a 01.2021.
[230] https://bbbc.broadinstitute.org/BBBC038. Acedido a 01.2021.
[231] https://www.kaggle.com/c/data-science-bowl-2018. Acedido a 01.2021.
[232] L. Roux *et al.*, "Mitosis detection in breast cancer histological images An ICPR 2012 contest," *J. Pathol. Inform.,* vol. 4, May. 2013, doi: 10.4103/2153-3539.112693.
[233] PyTorch. https://pytorch.org/docs/stable/generated/torch.nn.Upsample.html. Acedido a 04.2021.
[234] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-net: Fully convolutional neural networks for volumetric medical image segmentation," in *2016 fourth international conference on 3D vision (3DV)*, Stanford, CA, USA, Oct. 2016: IEEE, pp. 565-571, doi: 10.1109/3DV.2016.79.
[235] X. Xia and B. Kulis, "W-net: A deep model for fully unsupervised image segmentation," *arXiv preprint arXiv:1711.08506,* Nov. 2017.
[236] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE international conference on computer vision*, Santiago, Chile, Dec. 2015, pp. 1520-1528, doi: 10.1109/ICCV.2015.178.
[237] J. Yang, R. Shi, and B. Ni, "MedMNIST Classification Decathlon: A Lightweight AutoML Benchmark for Medical Image Analysis," *arXiv preprint arXiv:2010.14925,* 2020.
[238] M. Hossin and M. Sulaiman, "A review on evaluation metrics for data classification evaluations," *Int. J. Data Min. Knowl. Manag. Process,* vol. 5, no. 2, p. 1, Mar. 2015, doi: 10.5121/ijdkp.2015.5201.
[239] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *arXiv preprint arXiv:1702.05659,* Feb. 2017.
[240] M. Z. Alom, C. Yakopcic, T. M. Taha, and V. K. Asari, "Nuclei segmentation with recurrent residual convolutional neural networks based U-Net (R2U-Net)," in *NAECON 2018-IEEE National Aerospace and Electronics Conference*, Dayton, OH, USA, Jul. 2018: IEEE, pp. 228-233, doi: 10.1109/NAECON.2018.8556686.